SISTEM OPERASI

Meyti Eka Apriyani
Elok Nur Hamdana
Rinanza Zulmy Alhamri

CERDAS ULET KREATIF
www.cerdas.co.id

Sistem Operasi

Penulis:
Meyti Eka Apriyani
Elok Nur Hamdana
Rinanza Zulmy Alhamri

Penyunting:
Dhega Febiharsa

Desain & Tata Letak:
Tim Penerbit Cerdas Ulet Kreatif

ISBN :
Cetakan Pertama : **2022**
Penerbit :
Cerdas Ulet Kreatif (Anggota IKAPI No. 156 / JTI / 2014)
Jl. Manggis 72 RT 03 RW 04 Jember Lor - Patrang
Jember - Jawa Timur 68118
Telp. 0331-4431347, 412387 Faks. 4431347
e-mail : info@cerdas.co.id

Distributor Tunggal:
Cerdas Ulet Kreatif
Jl. Manggis 72 RT 03 RW 04 Jember Lor - Patrang
Jember - Jawa Timur 68118
Telp. 0331-4431347, 412387 Faks. 4431347
e-mail : info@cerdas.co.id

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Allah SWT, karena hanya dengan rahmad hidayah serta inayahNya sehingga buku yang berjudul “Sistem Operasi” dapat terselesaikan dengan baik. Buku ini disusun dan dibuat dengan tujuan agar dapat menambah pengetahuan dan wawasan dalam belajar sehingga dapat memahami dasar dasar pemrograman framework dengan baik.

Buku ini membantu memahami tentang Sistem Operasi dalam Sistem Komputer. Buku ini lebih khusus diperuntukkan kepada para mahasiswa yang sedang mengambil mata kuliah Sistem Operasi dan dasarnya. Oleh karena itu beberapa contoh dan Latihan yang ada pada buku ini sebagian diambil dari dunia kemahasiswaan. Mudah mudahan dengan mempelajari buku ini, para pembaca akan mampu menghadapi masalah atau kesulitan yang timbul dalam pembelajaran dan dengan harapan semoga mampu berinovasi dan berkreasi dengan potensi yang dimiliki.

Kami menyadari bahwa dalam pembuatan buku ini masih terdapat kekurangan sehingga penulis berharap saran dan kritik dari pembaca agar penulis dapat meningkatkan dan memperbaiki penyajian materi yang lebih baik dari sebelumnya. Akhir kata penulis ucapkan terima kasih

Malang, Juni 2022

Penulis

DAFTAR ISI

DAFTAR GAMBAR

BAB 1
SISTEM OPERASI

Sistem operasi adalah perangkat lunak (*Software*) yang dapat melakukan tugas mengontrol dan mengatur perangkat keras (*hardware*) sekaligus operasi dasar sistem lainnya. Sistem operasi dapat digunakan untuk menjalankan program aplikasi dan sebagai perantara antara *user* dan perangkat keras (*hardware*). Sistem operasi *mainframe* didesain untuk mengoptimalkan manfaat dari *hardware*. Sistem operasi yang ada pada komputer pribadi (PC) sangat mendukung untuk game yang kompleks dan aplikasi bisnis. Sistem operasi yang ada pada komputer genggam dirancang untuk memudahkan pengguna di manapun dia berada agar mudah untuk berinteraksi dengan komputer dan menjalankan program. Beberapa sistem operasi dirancang agar nyaman dan efisien. Dalam bab ini akan memberikan gambaran umum tentang komponen utama dari sistem operasi tersebut.

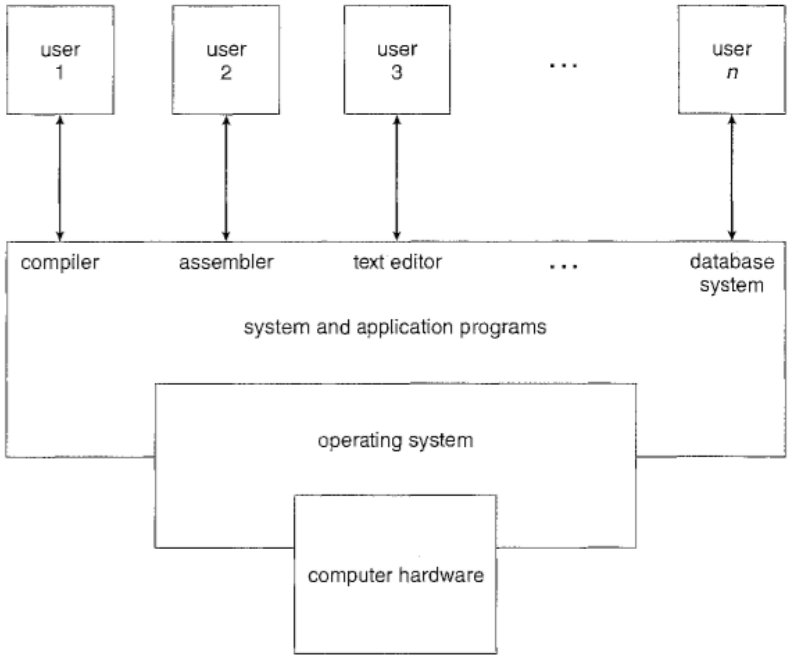
Capaian:

- Mampu memahami Komponen dari Sistem Operasi
- Mampu memahami organisasi Sistem operasi

A. Pengertian Sistem Operasi

Secara umum sistem komputer terdiri dari empat bagian yaitu; *hardware*, sistem operasi, program aplikasi dan *user*. hardware, CPU, memori, dan perangkat *input output* (I/O) merupakan penyedia sumber daya komputasi yang dasar untuk sistem. Program aplikasi seperti *msword*, *msexel compiler*, dan *web browser* dapat digunakan untuk menyelesaikan masalah yang berkaitan dengan komputasi. Sistem operasi mengontrol *hardware* dan mengoordinasi penggunaannya di antara program aplikasi yang bervariasi untuk user yang bervariasi pula. Sistem operasi menyediakan sarana untuk user dari sumber daya dalam pengoperasian

sistem komputer. Berikut ini gambaran hubungan *hardware*, sistem operasi dan aplikasi program.



Gambar 1 Komponen dari sistem komputer

Sudut pandang seorang pengguna komputer (*user*) sangat bervariasi disesuaikan dengan *interface* yang digunakan. Sebagian besar seorang *user* berada di depan PC, yang terdiri dari monitor, *keyboard*, *mouse*, dan unit sistem. Sistem seperti itu dirancang untuk satu pengguna untuk dapat menggunakan semua sumber dayanya. Tujuannya adalah untuk memaksimalkan pekerjaan yang dilakukan dan sistem operasi dirancang agar mudah untuk digunakan.

Peranan sistem operasi jika dilihat dari sisi *user* dan sistem. Bagaimana mendefinisikan apa itu sistem operasi? Secara umum, Sistem operasi ada karena menawarkan cara yang masuk akal untuk menyelesaikan suatu masalah dari sistem komputer. Tujuan mendasar dari sistem komputer adalah untuk menjalankan program dari user dan memecahkan masalah agar lebih mudah. Untuk mencapai tujuan ini maka dibangunlah sebuah *hardware* yang mana program-program

ini memerlukan suatu operasi tertentu, seperti mengendalikan perangkat I/O. Fungsi umum untuk mengontrol dan mengalokasikan sumber daya yang kemudian disatukan menjadi satu perangkat lunak yang disebut dengan sistem operasi.

B. Kegunaan Sistem Operasi

Sistem operasi berfungsi sebagai pengelola perangkat lunak (*software*) dan perangkat keras (*hardware*) pada sebuah komputer, dengan kata lain sistem operasi dapat mengkondisikan komputer agar dapat menjalankan program atau aplikasi sesuai dengan keinginan dari *user*. Sistem operasi juga berfungsi sebagai pengontrol masalah *user*, Untuk menghindari masalah yang terjadi pada saat user menggunakan sumber daya yang sama, sistem operasi mengatur user mana yang dapat mengakses suatu sumber daya. Sistem operasi juga sering disebut *resource allocator*. Selain itu fungsi penting dari sistem operasi adalah sebagai program pengendali yang bertujuan untuk menghindari kekeliruan (*error*) dan penggunaan komputer yang tidak perlu. Fungsi sistem operasi secara umum adalah sebagai berikut:

1 *Menjalankan Operasi Dasar*

Fungsi utama dari sistem operasi adalah menjalankan operasi dasar pada komputer. Sistem ini dinilai sebagai komponen vital yang mendasari kerja perangkat lunak atau software lainnya.

Sebelum aplikasi berjalan dan dapat berfungsi pada suatu komputer, maka sistem operasilah yang memungkinkan program atau aplikasi tersebut dapat berjalan dan ditampilkan kepada *user* yang menggunakan perangkat komputer tersebut.

2 *Mengatur Kerja Hardware Dan Software*

Sistem operasi adalah *Resource Manager* dalam perangkat komputer. Maksudnya, sistem operasi berfungsi mengontrol fungsi perangkat keras seperti memori, CPU, *harddisk*, dan

perangkat keras lainnya dan juga mengatur fungsi program software agar terhubung dengan perangkat keras tersebut.

3 *Wadah Program Atau Aplikasi*

Aplikasi-aplikasi yang ada dikomputer memang tersimpan dalam perangkat penyimpanan akan tetapi sebenarnya aplikasi atau program tersebut berada dalam wadah yang merupakan sistem operasi itu sendiri. Aplikasi tersebut melekat pada sistem operasi dan tidak dapat berfungsi tanpa adanya sistem operasi tersebut.

4 *Menyajikan Tampilan*

Tampilan yang dilihat dilayar komputer maupun gadget adalah hasil dari pengoperasian OS atau sistem operasi. Dengan kata lain, sistem operasi memudahkan aplikasi berjalan sekaligus menampilkannya pada monitor layar komputer atau menterjemahkan bahasa pemrograman yang berasal dari CPU kemudian menampilkannya dalam bentuk teks, grafis dan tampilan lainnya yang mudah dimengerti.

5 *Mengkoordinasi Kerja Perangkat Komputer*

Selain mengatur perangkat keras dan perangkat lunak agar terhubung satu sama lain, sistem operasi juga berfungsi mengkoordinasikan segala hal didalam komputer terutama menyusun program yang sifatnya kompleks menjadi lebih sederhana dan berurutan. Sistem operasi memudahkan suatu aplikasi agar dapat bekerja dengan lebih efisien.

6 *Mengoptimalkan Fungsi Perangkat Komputer*

Fungsi lain dari sistem operasi atau OS adalah mengoptimalkan penggunaan perangkat keras maupun perangkat lunak. Misalnya dalam hal mengatur waktu berfungsinya CPU, pemanggilan data yang tersimpan dalam memori harddisk, ataupun mengatur waktu yang digunakan untuk koneksi dalam terminal.

C. Sejarah Sistem Operasi

Studi tentang sistem operasi dipermudah dengan ketersediaan sejumlah sistem operasi *open-source* yang tersedia dalam format *source code* sebagai kode biner yang telah dikompilasi. *linux* merupakan sistem operasi *open-source* yang paling terkenal, sementara *microsoft windows* merupakan contoh terkenal dari kebalikannya. Dimulai dengan adanya sebuah *source code* memungkinkan seorang *programmer* untuk menghasilkan kode biner yang dapat dieksekusi pada suatu sistem. Banyak sekali manfaat dari sistem operasi yang open source termasuk komitmen seorang *programmer* yang berkontribusi dalam kode dengan membantu men-*debug* dan menganalisisnya serta memberikan dukungan dan bahkan memberikan masukan.

Pada awal komputasi modern ada (tahun 1950-an), banyak perangkat lunak yang tersedia dalam format *open source*. Seorang hacker di MIT's Tech Model Railroad Club meninggalkan program mereka di laci untuk dikerjakan orang lain. Grup pengguna "Homebrew" bertukar kode selama pertemuan mereka. Kemudian, grup pengguna khusus perusahaan, seperti Peralatan Digital DEC Perusahaan, memberikan kontribusi yang diterima dari program berupa *source code*. Perusahaan komputer dan perangkat lunak akhirnya berusaha membatasi penggunaan perangkat lunak mereka ke komputer resmi dan pelanggan yang membayar. Hanya memberika file yang sudah dikompilasi dari *source code* untuk membantu mereka mencapai tujuan ini, serta melindungi kode mereka dan ide-ide mereka dari pesaing mereka. Masalah lain yang melibatkan hak cipta. Sistem operasi dan program lain dapat membatasi kemampuan untuk bermain mengembalikan film dan musik atau menampilkan buku elektronik ke komputer resmi.

Hukum di banyak negara, termasuk A.S. Hak Cipta Milenium Digital Act (DMCA), membuatnya ilegal untuk merekayasa balik kode DRM atau mencoba untuk menghindari perlindungan salinan.

Untuk melawan langkah membatasi penggunaan dan redistribusi perangkat lunak, Richard Stallman pada tahun 1983 memulai proyek GNU untuk membuat UNIX sistem

operasi yang kompatibel. Pada tahun 1985, ia menerbitkan Manifesto GNU, yang berpendapat bahwa semua perangkat lunak harus bebas dan open source. Dia juga membentuk dengan tujuan mendorong yang free melakukan pertukaran source code dari perangkat lunak dan penggunaan gratis perangkat lunak tersebut. Daripada hak cipta perangkat lunaknya, FSF "menyalin" perangkat lunak untuk mendorong berbagi dan perbaikan.

1 Linux

Salah satu contoh sistem operasi yang *open source* adalah GNU/Linux. Proyek GNU menghasilkan banyak alat yang kompatibel dengan UNIX, termasuk kompiler, editor, dan utilitas, tetapi tidak pernah merilis kernel. Pada tahun 1991, seorang mahasiswa di Finlandia, Linus Torvalds, merilis kernel mirip UNIX yang belum sempurna menggunakan Kompiler dan alat GNU. Dengan adanya internet mempermudah siapa pun yang tertarik dapat mengunduh *souce code*, memodifikasi, dan mengirimkan perubahan ke Torvalds. Dengan adanya pembaruan di setiap minggunya memungkinkan sistem operasi linux ini berkembang pesat dan ditingkatkan oleh ribuan programmer yang ada di seluruh dunia.

2 BSD Unix

BSD Unix merupakan sejarah yang lebih panjang dan lebih rumit daripada Linux. Pada tahun 1978 ada sebagai turunan dari AT&T's UNIX. Dirilis oleh University of California di Kota Berkeley (UCB) dalam bentuk *source* dan *biner*, tetapi belum *open source* karena memerlukan lisensi dari AT&T. Pengembangan BSD UNIX diperlambat oleh gugatan oleh AT&T, yang akhirnya menjadi *open source*. Versi dari BSD Unix diantaranya yaitu, 4.4BSD-lite, dirilis pada tahun 1994. Sama seperti Lim.IX, ada banyak distro BSD UNIX, antara lain FreeBSD, NetBSD, OpenBSD, dan DragonflyBSD.

3 Solaris

Solaris adalah sistem operasi berbasis UNIX komersil. atau Sun Microsystem. Awalnya, sistem operasi Sun didasarkan pada BSD UNIX. Sun pindah ke Sistem V UNIX AT&T sebagai

basisnya pada tahun 1991. Pada tahun 2005, Sun melakukan open source beberapa kode Solaris, dan seiring berjalannya waktu, perusahaan menambahkan lebih banyak lagi ke basis kode yang open source. Sayangnya, tidak semua Solaris bersifat *open source*, karena beberapa kode masih dimiliki oleh AT&T dan perusahaan lain. Namun, Solaris dapat dikompilasi dari *open source* dan ditautkan dengan binari dari komponen yang bersumber dari dekat, sehingga masih dapat dieksplorasi, dimodifikasi, dikompilasi, dan diuji. *Source code* terdapat pada alamat http: // opensolaris. org/ os/ downloads/.

D. RINGKASAN

Agar komputer dapat melakukan tugasnya dalam mengeksekusi program, program tersebut harus terdapat dalam memori utama. Memori utama adalah satu-satunya area penyimpanan besar yang digunakan prosesor dapat mengakses secara langsung. Susunan kata dalam byte, dengan ukuran mulai dari jutaan hingga miliaran. Setiap kata dalam memori memiliki alamatnya sendiri. Memori utama biasanya merupakan perangkat penyimpanan yang mudah menguap yang kehilangan isinya saat daya dihidupkan, mati atau hilang. Sebagian besar sistem komputer menyediakan penyimpanan sekunder sebagai ekstensi atau memori utama. Penyimpanan sekunder menyediakan bentuk penyimpanan yang mampu menyimpan data dalam jumlah besar secara permanen. Yang paling umum perangkat penyimpanan sekunder adalah disk magnetik, yang menyediakan penyimpanan keduanya program dan data. Sistem operasi harus memastikan pengoperasian komputer yang benar. Untuk mencegah program pengguna mengganggu pengoperasian yang benar dari sistem, perangkat keras memiliki dua mode: mode pengguna dan mode kernel. Berbagai instruksi (seperti instruksi I/0 dan instruksi berhenti) memiliki hak istimewa dan hanya dapat dijalankan dalam mode kernel. Memori di mana operasi sistem berada juga harus dilindungi dari modifikasi oleh pengguna. Sebuah kaleng1.er mencegah *loop* tak terbatas. Fasilitas ini (mode ganda, instruksi istimewa, perlindungan memori, dan interupsi

timer) adalah blok bangunan dasar yang digunakan oleh sistem operasi untuk mencapai operasi yang benar.

E. JAWABAN DAN SOAL

1. Jelaskan pengertian dari Sistem Operasi!
   Sistem operasi adalah perangkat lunak yang mengelola perangkat keras komputer, juga sebagai lingkungan untuk menjalankan program aplikasi. Mungkin aspek yang paling terlihat dari sistem operasi adalah antarmuka ke komputer sistem yang diberikannya kepada pengguna manusia
2. Jelaskan secara singkat histori sistem operasi!
   GNU /Linux, BSD UNIX, dan Solaris semuanya adalah sistem operasi sumber terbuka. Keuntungan dari perangkat lunak bebas dan sumber terbuka cenderung meningkatkan jumlah dan kualitas proyek sumber terbuka, yang mengarah pada peningkatan jumlah individu dan perusahaan

BAB 2
STRUKTUR SISTEM OPERASI

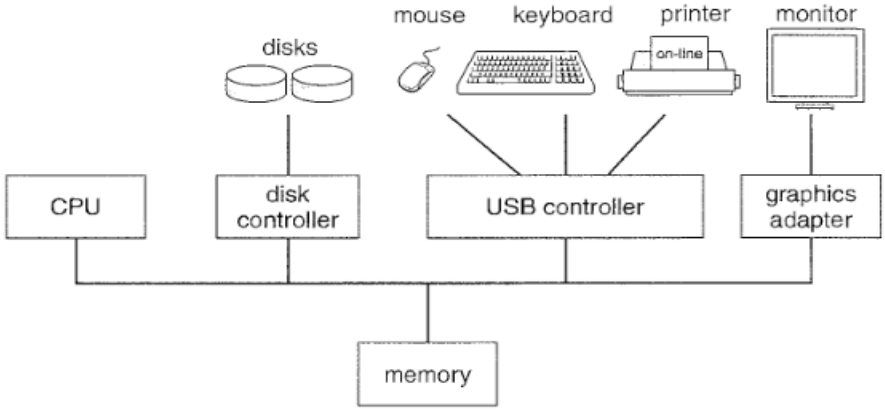
Sistem operasi menyediakan lingkungan di mana program-program dieksekusi. Secara internal, sistem operasi sangat bervariasi dalam susunannya, karena diatur di sepanjang garis yang berbeda. Perancangan sistem operasi baru merupakan pekerjaan utamanya. Hal yang penting sebelum membuat perancangan adalah dengan menfinisikan sistem itu sendiri. Tujuan ini menjadi dasar untuk pemilihan algoritma dan strategi

Capaian:

1. Untuk mendeskripsikan komponen dari sistem operasi
2. Untuk Menjelaskan struktur dari sistem operasi

A. Komponen Sistem Operasi

Sistem komputer serba guna modern terdiri dari satu atau lebih CPU dan sejumlah *device controller* yang terhubung melalui secara umum yang menyediakan akses ke memori bersama (Gambar 2). Setiap *device controller* ada di pengisian daya jenis perangkat tertentu (misalnya, disk drive, perangkat audio, dan tampilan video). CPU dan *device controller* dapat melakukan eksekusi secara bersamaan, bersaing untuk siklus memori. Untuk memastikan akses yang teratur ke memori bersama, disediakan pengontrol memori yang fungsinya untuk mensinkronkan akses ke memori.



Gambar 2 Sistem Komputer Modern

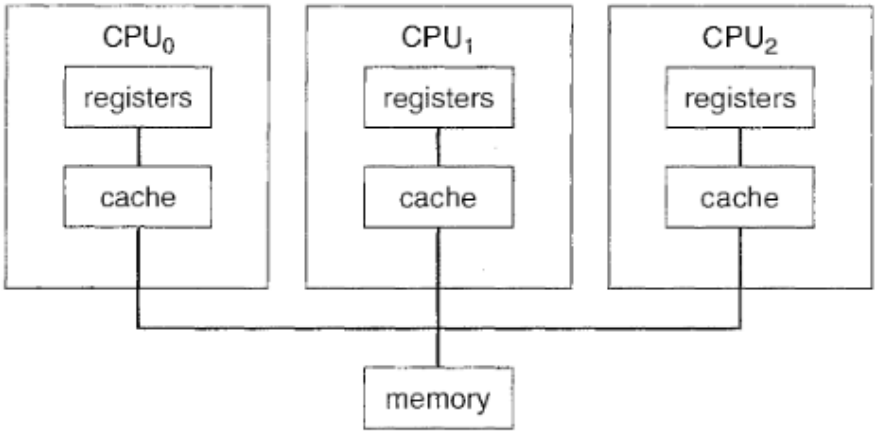
Agar komputer dapat mulai bekerja (saat dihidupkan atau reboot) perlu memiliki program awal untuk dijalankan. Program awal atau *bootstrap program* ini cenderung sederhana. Biasanya, disimpan dalam *read only memory* (ROM) atau *electrically erasable programmable read only memory* (EEPROM) yang dikenal dengan istilah umum sebagai *firmware* di dalam perangkat keras komputer. Hal ini menginisialisasi semua aspek pada sistem, dari register CPU ke *device controller* kemudian ke isi memori. *bootstrap program* harus tahu cara memuat sistem operasi dan bagaimana memulai menjalankan sistem itu. Untuk menyelesaikan ini tujuannya, *bootstrap program* harus mencari dan memuat ke dalam memori operasi inti dari sistem. Sistem operasi kemudian mulai menjalankan proses pertama, seperti "init", dan menunggu beberapa peristiwa terjadi.

B. Struktur Penyimpanan

Penggunaan komputer merupakan bentuk lain dari memori. Karena *read-only memory* (ROM) tidak dapat diubah, hanya program statis yang disimpan di sana. Kekekalan ROM digunakan dalam kartrid game. EEPROM tidak dapat sering diganti dan sebagian besar berisi program statis. Misalnya, *smartphone* memiliki EEPROM untuk menyimpan program mereka yang tidak berjalan di pabrik. Idealnya, program dan data yang berada dimemori utama permanen. Pengaturan ini biasanya tidak memungkinkan untuk dua hal berikut ini:

1. Memori utama biasanya terlalu kecil untuk menyimpan semua program dan data yang dibutuhkan secara permanen.
2. Memori utama adalah perangkat penyimpanan yang mudah menguap yang kehilangan isinya ketika listrik dimatikan atau hilang.

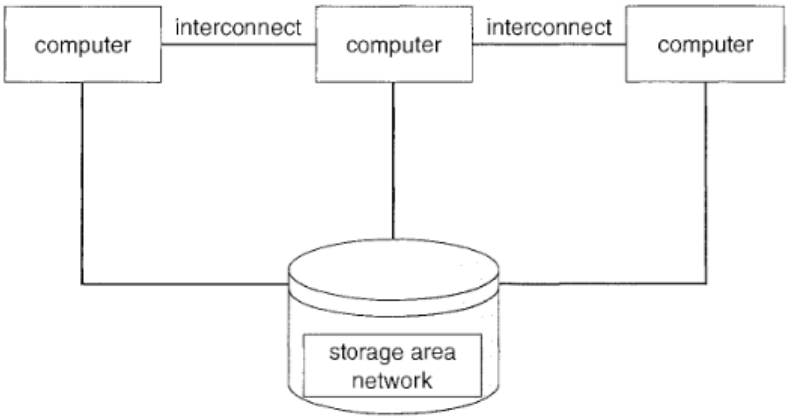
Sebagian besar sistem komputer menyediakan *secondary storage* sebagai ekstensi dari memori utama. Persyaratan utama untuk *secondary storage* adalah dapat menyimpan data dalam jumlah besar secara permanen. Perangkat *secondary storage* yang paling umum adalah disk magnetik yang menyediakan penyimpanan untuk program dan data. Sebagian besar program (sistem dan aplikasi) disimpan pada disk sampai dimuat ke dalam memori. Banyak program kemudian menggunakan disk sebagai sumber dan tujuan mereka untuk pengolahan. Oleh karena itu, manajemen penyimpanan disk yang tepat sangat penting untuk sistem komputer. Namun, dalam arti yang lebih besar, struktur penyimpanan yang terdiri dari register, memori utama, dan disk magnetik hanyalah salah satu dari banyak sistem penyimpanan yang mungkin. Lainnya termasuk memori *cache*, CD-ROM, magnetic kaset, dan sebagainya. Setiap sistem penyimpanan menyediakan fungsi dasar penyimpanan datum dan memegang datum itu sampai diambil di lain waktu. Perbedaan Utama di antara berbagai sistem penyimpanan terletak pada kecepatan, biaya, dan ukuran. Berbagai macam sistem penyimpanan dalam sistem komputer dapat diatur dalam Gambar 3 menurut kecepatan dan biaya. Level yang lebih tinggi adalah mahal, hierarki, biaya per bit umumnya menurun, sedangkan waktu akses umumnya meningkat. Jika sistem penyimpanan yang diberikan lebih cepat dan lebih murah daripada yang lain maka tidak akan ada alasan untuk menggunakan memori yang lebih lambat dan lebih mahal. Faktanya, banyak penyimpanan awal perangkat, termasuk pita kertas dan memori inti. Empat tingkat memori teratas pada Gambar 3 dapat dibangun menggunakan *semiconductor memory.*



Gambar 3 Arsitektur multiprosesing

C. Sistem Kluster

Jenis lain dari sistem multi-CPU adalah sistem cluster Seperti sistem multiprosesor, sistem cluster mengumpulkan beberapa CPU untuk menyelesaikan pekerjaan komputasi. Sistem cluster berbeda dari sistem multiprosesor namun, karena mereka terdiri dari dua atau lebih sistem individu yang bergabung bersama. Definisi atau istilah berkerumun yang tidak konkrit maka banyak persepsi tentang apa itu sistem berkerumun dan mengapa satu bentuk lebih baik dari yang lain. Definisi yang diterima secara umum adalah bahwa komputer berbagi penyimpanan dan terkait erat melalui *local area network* (LAN) atau interkoneksi yang lebih cepat, seperti *InfiniBand*.

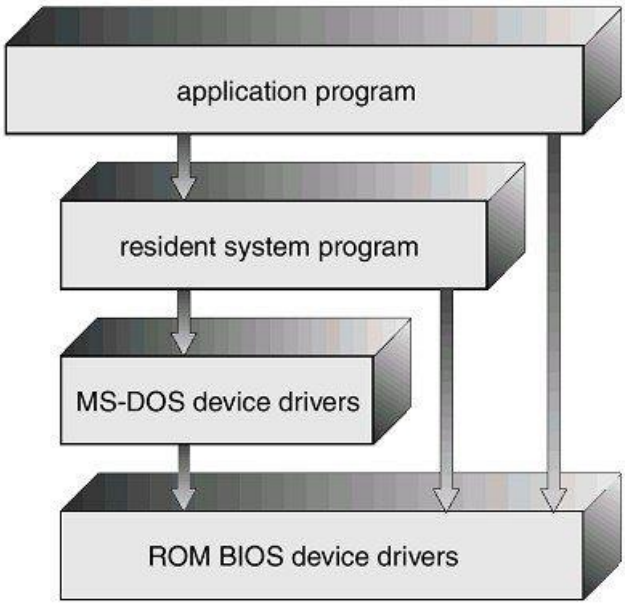


Gambar 4 Struktur umum sistem kluster

Teknologi kluster berubah dengan cepat. Beberapa produk klaster mendukung puluhan sistem dalam sebuah kluster, serta node berkerumun yang dipisahkan dengan mil. Banyak dari peningkatan ini dimungkinkan oleh *store area network* (SANs) yang memungkinkan banyak sistem untuk dilampirkan ke dalam penyimpanan. Jika aplikasi dan datanya disimpan di SAN, maka perangkat lunak klaster dapat menetapkan aplikasi untuk dijalankan pada semua *host* yang dilampirkan ke SAN.

D. Struktur Sistem Operasi

Terdapat sistem komersil yang tidak memiliki struktur yang cukup baik. Sistem operasi tersebut sangat kecil, sederhana dan memiliki banyak ketebatasan. Salah satu contoh sistem tesebut adalah MS DOS dirancang oleh orang-orang yang tidak memikirkan akan kepopuleran software tersebut. Sistem operasi tersebut terbatas pada hardware sehingga tidak terbagi bagi menjadi modul-modul seperti terlihat pada gambar 5. Karena Intel 8088 tidak menggunakan *dual mode* sehingga tidak ada proteksi *hardware*.



Gambar 5 Struktur sistem operasi

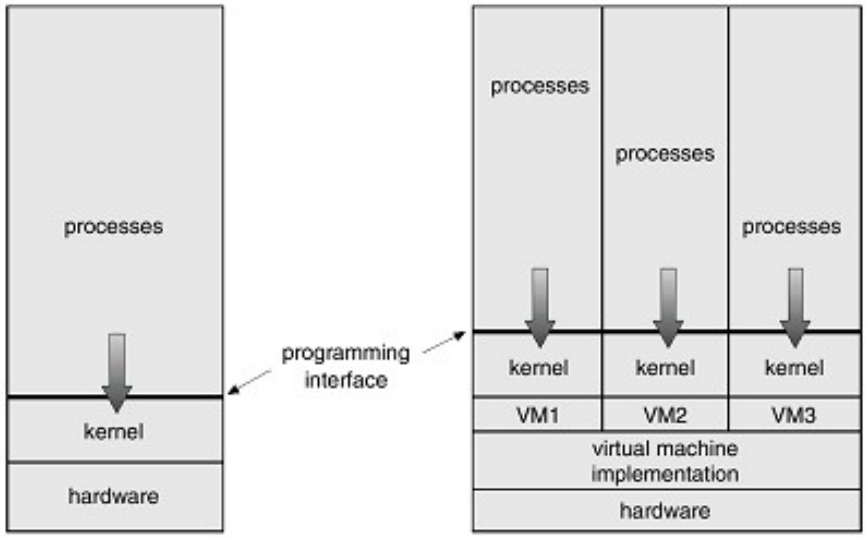
Pada dasarnya, sistem *monolitic* merupakan struktur sederhana yang dilengkapi dengan operasi *dual mode*. Pelayanan (*system calls*) yang diberikan oleh sistem operasi model ini dilakukan dengan cara mengambil sejumlah parameter pada tempat yang telah ditentukan sebelumnya, seperti register dan kemudian mengeksekusi suatu instruksi trap tertentu pada monitor mode.

Secara umum *system calls* dibuat dengan cara:

1. *User* program melakukan '*trap'* pada kernel. Instruksi berpindah dari *mode user* ke *mode monitor* dan mentransfer kontrol ke sistem operasi
2. Sistem operasi mengecek parameter-parameter dari pemanggilan tersebut untuk menentukan *system call* mana yang memanggil
3. Sistem operasi menunjuk ke suatu tabel yang berisi slot ke –k yang menunjukkan *system call*
4. Setelah *system call* selesai mengerjakan tugasnya, konrol akan dikembalikan pada *user* program.

Teknik pendekatan berlapis pada dasarnya dibuat dengan cara membentuk sistem operasi menjadi bentuk modular.

Dengan menggunakan pendekatan *top-down*, semua fungsi ditentukan dan dibagi menjadi komponen-komponen. Modularisasi sistem dilakukan denga cara memecah sistem operasi menajdi beberapa lapis (tingkat). Lapisan terendah (lapis-0) adalah *hardware* dan lapisan teratas (lapisan N) adalah *user interface*.



Gambar 6 Struktur sistem berlapis

Pada sebuah mesin virtual (*Virtual Machine*) misalkan terdapat sistem program => control program yang mengatur pemakaian sumber daya perangkat keras.
Control program = *trap System call* + akses ke perangkat keras.
Control program memberikan fasilitas ke proses pengguna. Mendapatkan jatah CPU dan memori. Menyediakan interface "identik" dengan apa yang disediakan oleh perangkat keras => sharing devices untuk berbagai proses.
Mesin Virtual (MV) => control program yang minimal MV memberikan ilusi multitasking: seolah-olah terdapat prosesor dan memori ekslusif digunakan MV. MV memilah fungsi multitasking dan implementasi *extended machine* (tergantung proses pengguna) => flexible dan lebih mudah untuk

pengaturan. Jika setiap pengguna diberikan satu MV => bebas untuk menjalankan OS (kernel) yang diinginkan pada MV tersebut. Potensi lebih dari satu OS dalam satu komputer. Contoh: IBM VM370: menyediakan MV untuk berbagai OS: CMS (interaktif), MVS, CICS, dll. Masalah: Sharing disk => OS mempunyai sistem berkas yang mungkin berbeda. IBM: virtual disk (minidisk) yang dialokasikan untuk pengguna melalui MV.
Konsep MV menyediakan proteksi yang lengkap untuk sumberdaya sistem, dikarenakan tiap MV terpisah dari MV yang lain. Namun, hal tersebut menyebabkan tidak adanya sharing sumber daya secara langsung. MV merupakan alat yang tepat untuk penelitian dan pengembangan sistem operasi. Konsep MV susah untuk diimplementasi sehubungan dengan usaha yang diperlukan untuk menyediakan duplikasi dari mesin utama.

E. RINGKASAN

- Sistem operasi terdiri dari beberapa komponen, antara lain manajemen proses, manajemen memori utama, manajemen file, manajemen sistem I/O, manajemen penyimpan sekunde r, system jaringan, system proteksi dan *system command interpreter*.
- Sistem komputer modern yang semakin komplek dan rumit memerlukan sistem operasi yang dirancang dengan sangat hati-hati agar dapat berfungsi secara optimum dan mudah untuk dimodifikasi

F. SOAL DAN JAWABAN

Jelaskan yang dimaksud dengan Sistem program!

Sistem program menyediakan lingkungan yang nyaman untuk pengembangan dan eksekusi program. Kebanyakan user melihat system operasi yang didefinisikan oleh system program dan bukan system call sebenarnya.

BAB 3
PROSES SIKLUS HIDUP

Sistem komputer awal hanya mengizinkan satu program untuk dieksekusi pada satu waktu. Program ini memiliki kendali penuh atas sistem dan memiliki akses ke semua sumber daya sistem. Sebaliknya, sistem komputer saat ini memungkinkan beberapa program untuk dimuat ke dalam memori dan dieksekusi secara bersamaan. Evolusi ini membutuhkan kontrol yang lebih kuat dan lebih banyak kompartementalisasi berbagai program dan kebutuhan ini menghasilkan gagasan tentang suatu proses/ yang sebuah program dalam eksekusi. Proses adalah unit kerja dalam pembagian waktu sistem modern. Semakin kompleks sistem operasi, semakin banyak yang diharapkan untuk dilakukan atas nama penggunanya. Meskipun perhatian utamanya adalah eksekusi program pengguna, itu juga perlu mengurus berbagai tugas sistem yang lebih baik ditinggalkan di luar kernelnya sendiri. Oleh karena itu, sebuah sistem terdiri dari kumpulan proses, proses operasi sistem yang mengeksekusi kode sistem dan proses pengguna yang mengeksekusi pengguna kode. Berpotensi untuk semua proses dapat dijalankan secara bersamaan dengan CPU digandakan di antara mereka. Dengan mengalihkan CPU antar proses, sistem operasi dapat membuat komputer lebih produktif.

Capaian:

1 Untuk memperkenalkan gagasan proses program dalam eksekusi, yang membentuk dasar dari semua perhitungan.
2 Untuk menggambarkan berbagai fitur proses, termasuk penjadwalan, penciptaan dan penghentian, dan komunikasi.
3 Untuk menggambarkan komunikasi dalam sistem *client-server*.

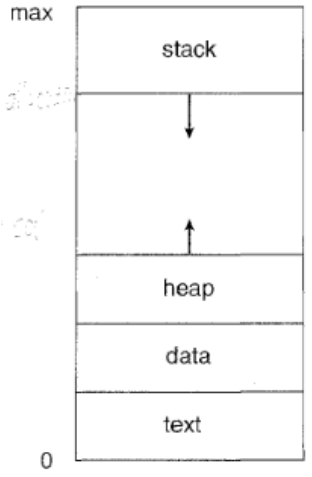
A. Pengertian Proses Siklus Hidup

Proses adalah unit kerja terkecil yang secara individu mempunyai sumber daya dan dijadwalkan oleh sistem operasi. Untuk kegiatan proses tersebut, sistem operasi berperan untuk mengelola segala proses pada sistem dan mengalokasikan sumber daya ke proses tersebut. Berbagai proses berjalan

secara bersamaan, dimana setiap proses mendapatkan bagian-bagian memori dan kendali sendiri-sendiri (peran SO), sehingga setiap proses (program) memiliki 2 prinsip di bawah ini:

1. *Independent*, program-program tersebut berdiri sendiri, tidak saling bergantung dan terpisah.
2. *One program at any instant*, hanya satu proses yang dilayani oleh pemroses pada satu waktu.

Secara informal, proses adalah program yang sedang dieksekusi. Sebuah proses lebih dari kode program, yang kadang-kadang dikenal sebagai bagian teks. Hal ini juga mencakup aktivitas saat ini, yang diwakili oleh nilai program *counter* dan isi *register prosesor*. Sebuah proses umumnya juga termasuk tumpukan proses, yang berisi data sementara (seperti fungsi parameter, alamat pengirim, dan variabel lokal), dan bagian data, yang berisi variabel global. Suatu proses juga dapat menyertakan *heap*, yaitu memori yang dialokasikan secara dinamis selama waktu proses berjalan. Struktur sebuah proses dalam memori ditunjukkan pada Gambar 7.



Gambar 7 Proses di memori

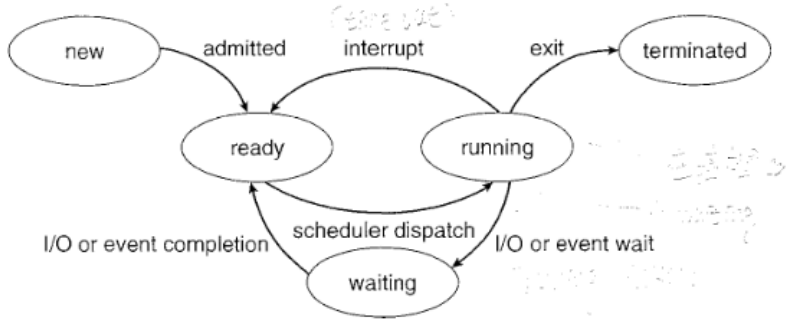
Program adalah pasif entitas, seperti file yang berisi daftar instruksi yang disimpan di disk (sering disebut file yang dapat dieksekusi), sedangkan proses adalah entitas aktif, dengan penghitung program menentukan instruksi berikutnya untuk dieksekusi dan satu set sumber daya terkait. Sebuah program

menjadi proses ketika file yang dapat dieksekusi dimuat ke dalam memori. Dua teknik umum untuk memuat file yang dapat dieksekusi adalah mengklik dua kali pada ikon yang mewakili file yang dapat dieksekusi dan memasukkan nama yang dapat dieksekusi file pada baris perintah (seperti dalam prog.exe atau a.out.)

Proses merupakan aktivitas/kondisi saat suatu program dilakukan eksekusi. Sebuah proses memerlukan berbagai sumber daya untuk penunjang dalam penyelesaian pekerjaan. Sumber daya ini dapat seperti memori, perangkat input output serta CPU *time*. (Permadi, M.Kom. & Vitadiar, S.SI,.M.Kom., 2022).

Proses adalah program yang sedang di jalankan. Proses harus berjalan secara berurutan (*sequential*). Sebuah proses terdiri dari program *counter* (PC), *stack* dan data *section*. (Mufadhol, 2015).

Suatu pengeksekusian proses dapat berubah-ubah keadaannya antara yang satu dengan lainnya seperti terdapat pada Gambar 8



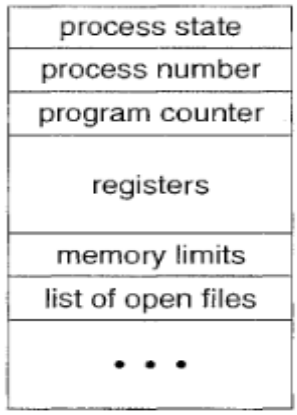
Gambar 8 Diagram proses state

Terdapat lima keadaan saat proses dijalankan antara lain:

1. *New* : Sebuah proses yang sedang dibuat
2. *Running* : Instruksi yang sedang dieksekusi
3. *Waiting* : Sebuah proses sedang menunggu untuk beberapa peristiwa dapat terjadi

4. *Ready* : Sebuah proses sedang menunggu untuk ditugaskan ke prosesor
5. *Terminated* : Sebuah proses telah selesai dieksekusi

Setiap proses memiliki informasi yang disimpan dalam blok kendali proses (PCB–*Process Control Block*). PCB berisi informasi tentang status proses, PC, register CPU, informasi penjadwalan CPU, informasi pengelolaan memori, informasi akunting dan informasi status masukan/keluaran.



Gambar 9 Proses Control Block (PCB)

1. *Process State*. Saat *state* berada pada kondisi *new*, *ready*, *running*, *waiting*, *halted*.
2. Program *Counter* Penghitung menunjukkan alamat instruksi berikutnya yag akan dieksekusi untuk proses ini.
3. CPU Register. Register bervariasi dalam jumlah dan jenis, tergantung pada arsitektur komputer. Termasuk akumulator, register indeks, stack pointer, dan register tujuan umum, ditambah kode kondisi apapun.
4. CPU *scheduling information*. Informasi ini mencakup prioritas proses, pointer ke antrian penjadwalan, dan parameter penjadwalan lainnya.
5. *Memory management information*. Informasi ini termasuk informasi seperti nilai register basis dan limit, tabel

halaman atau tabel segmen, tergantung pada sistem memori yang digunakan oleh sistem operasi

6. *Accounting Information*. Formasi ini termasuk jumlah CPU dan waktu nyata yang digunakan, batas waktu, nomor pekerjaan atau proses, dan seterusnya.
7. status informasi I/O. Informasi ini termasuk daftar perangkat I/O dialokasikan untuk proses, daftar file yang terbuka, dan sebagainya

Sebagian besar file hanya merupakan file biasa yang disebut file regular yang berisi data biasa. Sebagai contoh file text, file *executable* atau program, input atau output dari program dan lainnya. Selain file biasa ada file-file khusus seperti berikut:

1. *Directories* : file yang berisi daftar dari file lain.
2. *Special files* : mekanisme yang digunakan untuk input dan output. Sebagian besar terdapat pada direktori /dev.
3. *Links* : Sistem untuk membuat file atau direktori dapat terlihat di banyak bagian dari pohon file sistem.
4. *Sockets* (Domain) : Jenis file khusus, mirip dengan soket TCP/IP, yang menyediakan jaringan antar proses yang terproteksi oleh file *system's access control*.
5. *Named pipes* : berfungsi kurang lebih seperti soket dan membentuk jalur untuk proses komunikasi.

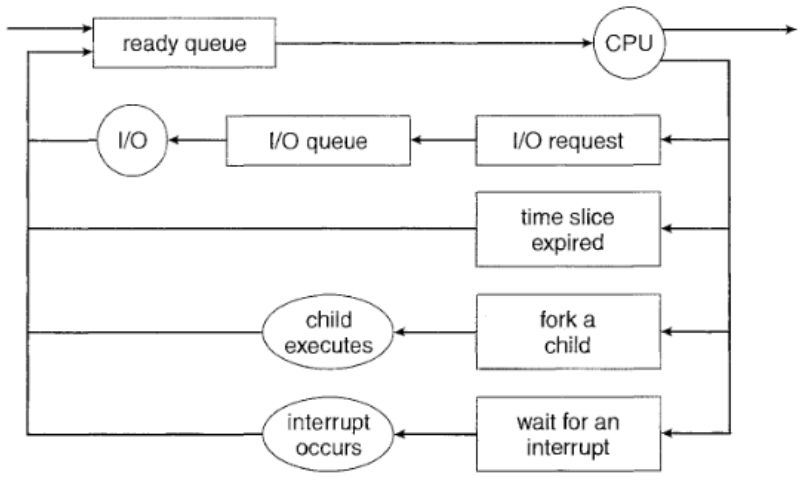
CPU dapat berpindah dari satu proses ke proses lainnya jika terjadi interupsi atau *system call*. Saat berpindah dari satu proses ke proses lain, CPU menyimpan informasi proses yang dihentikan ke PCB proses tersebut dan membaca PCB proses yang akan dieksekusi. Hal ini disebut dengan peralihan konteks (*context switch*). Peralihan konteks menyebabkan *overhead*, karena CPU tidak melakukan pekerjaan yang berguna pada saat peralihan tersebut. Selain itu, waktu yang dibutuhkan untuk melakukan peralihan konteks sangat bergantung pada perangkat keras (*hardware*).

Proses-proses memerlukan penjadwalan agar dapat memperoleh siklus CPU. Proses yang belum selesai terjadi karena beberapa hal, seperti menunggu layanan

masukan/keluaran, batas waktu yang diberikan sudah habis, memanggil anak proses, atau menunggu terjadinya interupsi. Sebelum dieksekusi di CPU, proses akan masuk ke dalam salah satu antrian:

1. Antrian Tugas (*job queue*) yaitu set semua proses yang ada didalam sistem
2. Antrian yang sudah siap (*ready queue*) yaitu set semua proses berada di memori utama, siap dan menunggu untuk eksekusi
3. Antrian Perangkat (*device or i/o queue*) yaitu proses - proses yang menunggu untuk perangkat I/O.

Suatu proses dapat membuat beberapa proses baru, melalui sistem pemanggilan pembuatan proses, selama jalur eksekusi. Pembuatan proses dinamakan induk proses, sebagaimana proses baru disebut anak dari proses tersebut. Tiap proses baru tersebut dapat membuat proses lainnya, membentuk suatu pohon proses. Secara umum, suatu proses akan memerlukan sumber tertentu (waktu CPU, memori, berkas, perangkat I/O) untuk menyelesaikan tugasnya. Ketika suatu proses membuat sebuah subproses, sehingga subproses dapat mampu untuk memperoleh sumbernya secara langsung dari sistem operasi. Induk mungkin harus membatasi sumber diantara anaknya, atau induk dapat berbagi sebagian sumber (seperti memori berkas) diantara beberapa dari anaknya. Membatasi suatu anak proses menjadi subset sumber daya induknya mencegah proses apa pun dari pengisian sistem yang telalu banyak dengan menciptakan terlalu banyak subproses. Sebagai tambahan pada berbagai sumber fisik dan logis bahwa suatu proses diperoleh ketika telah dibuat, data pemula (masukan) dapat turut lewat oleh induk proses sampai anak proses. Sebagai contoh, anggap suatu proses yang fungsinya untuk menunjukkan status sebuah berkas, katakan F1, pada layar terminal.



Gambar 10 Diagram representasi

Ketika dibuat, akan menjadi sebagai sebuah masukan dari proses induknya, nama dari berkas F1, dan akan mengeksekusi menggunakan kumpulan data tersebut untuk memperoleh informasi yang diinginkan. Proses tersebut juga mendapat nama dari perangkat luar. Sebagian sistem operasi melewati sumber-sumber ke anak proses. Pada sistem tersebut, proses baru dapat mendapat dua berkas terbuka yang baru, F1 dan perangkat terminal dan hanya perlu untuk mentransfer data antara kedua berkas tersebut.

Untuk mengilustrasikan implementasi yang berbeda ini, mari mempelajari sistem operasi UNIX. Dalam UNIX, tiap proses diidentifikasi oleh pengidentifikasi proses, yang merupakan integer yang unik. Proses baru dibuat oleh sistem pemanggilan *fork system call*. Proses baru tersebut terdiri dari sebuah copy ruang alamat dari proses aslinya (original). Mekanisme tersebut memungkinkan induk proses untuk berkomunikasi dengan mudah dengan anak proses. Kedua proses (induk dan anak) meneruskan eksekusi pada instruksi setelah *fork* dengan satu perbedaan: Kode kembali untuk *fork* adalah nol untuk proses baru (anak), sebagaimana proses pengidentifikasi non nol (*nonzero*) dari anak dikembalikan kepada induk.
Penjadwalan Proses:

1. *Long term scheduling*
    a. Penjadwalan jangka panjang, terjadi saat proses berada pada status *new* untuk dijadwalkan menjadi status *ready*
    b. *Long term scheduler* juga disebut *ready queue*
    c. Pemindahan proses dari memori skunder menuju ke memori utama
2. *Medium term scheduling*
    a. Penjadwalan jangka menengah, terjadi saat proses berada pada status *waiting* untuk dijadwalkan menjadi status *ready* kembali
    b. *Middle term scheduler* juga disebut *wait queue*
    c. Pemindahan proses berada pada internal memori utama
3. *Short term scheduling*
    a. Penjadwalan jangka pendek, terjadi saat proses berada pada status *ready* untuk dijadwalkan menjadi status *running*
    b. *Short term scheduler* juga disebut *CPU scheduler*
    c. Pemindahan proses dari memori utama menuju ke CPU.

Sebuah proses berakhir ketika proses tersebut selesai mengeksekusi pernyataan akhirnya dan meminta sistem operasi untuk menghapusnya dengan menggunakan sistem pemanggilan *exit*. Pada titik itu, proses tersebut dapat mengembalikan data (keluaran) pada induk prosesnya (melalui sistem pemanggilan *wait*). Seluruh sumber-sumber dari proses-termasuk memori fisik dan virtual, membuka berkas, dan penyimpanan I/O di tempatkan kembali oleh sistem operasi.

Ada situasi tambahan tertentu ketika terminasi terjadi. Sebuah proses dapat menyebabkan terminasi dari proses lain melalui sistem pemanggilan yang tepat (contoh *abort*). Biasanya, sistem seperti itu dapat dipanggil hanya oleh induk proses tersebut yang akan diterminasi. Bila tidak, pengguna dapat secara sewenang-wenang membunuh *job* antara satu sama lain. Catat bahwa induk perlu tahu identitas dari anaknya. Maka, ketika satu proses membuat proses baru, identitas dari proses yang baru diberikan kepada induknya.

Proses yang bersifat simultan (*concurrent*) dijalankan pada sistem operasi dapat dibedakaan menjadi yaitu proses *independent* dan proses kooperatif. Suatu proses dikatakan *independent* apabila proses tersebut tidak dapat terpengaruh atau dipengaruhi oleh proses lain yang sedang dijalankan pada sistem. Berarti, semua proses yang tidak membagi data apa pun (baik sementara/ tetap) dengan proses lain adalah *independent*.

Sedangkan proses kooperatif adalah proses yang dapat dipengaruhi atau pun terpengaruhi oleh proses lain yang sedang dijalankan dalam sistem. Dengan kata lain, proses dikatakan kooperatif bila proses dapat membagi datanya dengan proses lain. Ada empat alasan untuk penyediaan sebuah lingkungan yang memperbolehkan terjadinya proses kooperatif:

1. Pembagian informasi: apabila beberapa pengguna dapat tertarik pada bagian informasi yang sama (sebagai contoh, sebuah berkas bersama), harus menyediakan sebuah lingkungan yang mengizinkan akses secara terus menerus ke tipe dari sumber-sumber tersebut.
2. Kecepatan penghitungan/komputasi: jika menginginkan sebuah tugas khusus untuk menjalankan lebih cepat, harus membagi hal tersebut ke dalam subtask, setiap bagian dari subtask akan dijalankan secara parallel dengan yang lainnya. Peningkatan kecepatan dapat dilakukan hanya jika komputer tersebut memiliki elemen-elemen pemrosesan ganda (seperti CPU atau jalur I/O).
3. Modularitas: mungkin ingin untuk membangun sebuah sistem pada sebuah model modular-modular, membagi fungsi sistem menjadi beberapa proses atau *threads*.
4. Kenyamanan: bahkan seorang pengguna individu mungkin memiliki banyak tugas untuk dikerjakan secara bersamaan pada satu waktu. Sebagai contoh, seorang pengguna dapat mengedit, memcetak, dan meng-compile secara paralel.

B. System Calls

System Call adalah penyedia antarmuka dari pelayanan-palayanan yang tersedia dengan Sistem Operasi. Umumnya *System Call* menggunakan bahasa C dan C++, meskipun

tugas-tugas seperti *hardware* yang harus diakses langsung, maka menggunakan bahasa *assembly*. *System call* adalah merupakan suatu kumpulan instruksi *extended* yang disediakan oleh sistem operasi yang berfungsi sebagai interface antara sistem operasi dengan program pemakai. *System call* (biasa di kenal sebagai "*syscall*") adalah sebuah instruksi, mirip dengan instruksi "*add*" atau "*jump*". Pada tingkat tinggi, sebuah *system call* adalah cara sebuah program pada level *user* untuk meminta pada sistem operasi untuk menjalankan sesuatu untuknya. Jika seorang *programmer* membutuhkkan untuk membaca dari sebuah file, akan menggunakan *system call* untuk meminta sistem operasi untuk membaca file tersebut. Cara *system call* bekerja yaitu pertama-tama, *user* program akan mensetup argumen untuk *system call*. Salah satu argumen adalah nomor *system call*. Perlu di catat bahwa semua ini dilakukan secara automatis oleh fungsi *library* kecuali jika menulis menggunakan bahasa *assembler*. Sesudah semua argumen di setup, program akan menjalankan instruksi "*system call*". Instruksi ini akan menyebabkan *exception*: *event* yang akan menyebabkan processor untuk *jump* ke satu *address* dan mulai menjalankan program/code di *address* tersebut. Instruksi di alamat yang baru akan menyimpan *state user* program, menentukan sistem call apa yang inginkan, kemudian *call fuction* tersebut di kernel yang mengimplementasikan *system call*, setelah selesai maka mengembalikan program state, dan kembali ke user program. Sebuah *system call* adalah salah satu cara agar *function* yang di definisikan dalam *device driver* untuk dapat di panggil.

Keuntungan dan kerugian menggunakan *system call* sama antarmuka untuk memanipulasi baik file dan perangkat adalah Setiap perangkat dapat diakses seolah-olah itu adalah file dalam file sistem. Karena sebagian besar penawaran kernel dengan perangkat melalui antarmuka file, relatif mudah untuk menambahkan *device driver* baru dengan menerapkan kode perangkat keras khusus untuk mendukung antarmuka file abstrak. Oleh karena itu, ini manfaat pengembangan baik kode program pengguna, yang dapat bewritten untuk mengakses perangkat dan file dalam samemanner, dan perangkat sopir

kode, yang dapat ditulis untuk mendukung API yang didefinisikan dengan baik. Kerugian itu dengan menggunakan antarmuka yang sama adalah bahwa mungkin akan sulit untuk menangkap fungsi peralatan tertentu dalam konteks akses file API, sehingga baik mengakibatkan hilangnya fungsi atau kerugian kinerja. Beberapa ini dapat diatasi dengan penggunaan operasi ioctl yang menyediakan antarmuka tujuan umum untuk proses untuk memanggil operasi pada perangkat. Tujuan *system call*:

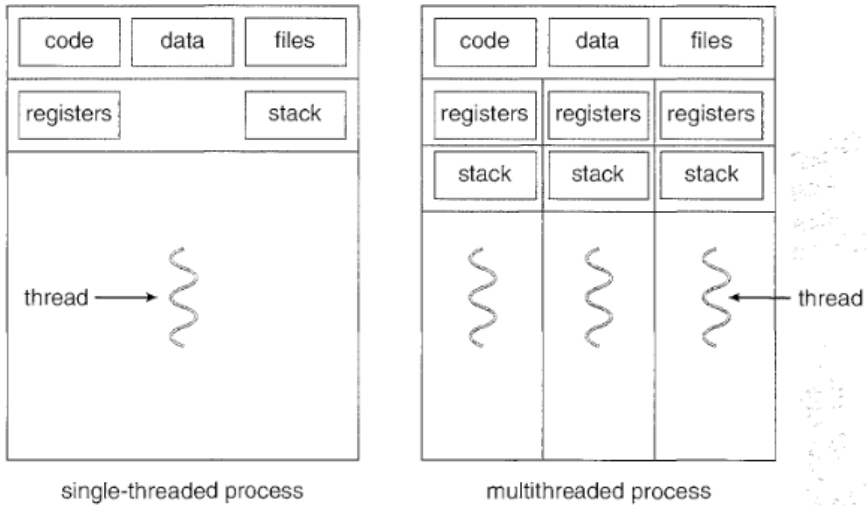
1. *System Calls For Signaling*, Pemanggilan sinyal interupsi yang digunakan untuk menghentikan suatu proses jika terdapat kesalahan atau jika ada proses lainnya yang perlu didahulukan.
2. *System Calls for File Management*, Sistem untuk manajemen file baik untuk membuat, membaca, membatasi pemakai memanipulasi file, dsb.

Ketika program mencoba untuk membuka file input dan ternyata tidak ada nama file itu atau file tersebut dilindungi pengaksesannya. Maka, harus membuat perintah di *command interpreter* yang terdapat di OS dan membukakan file tersebut. Jika file input ada, maka harus membuat file output baru. mungkin akan menemukan file output dengan nama yang sama. Situasi tersebut dapat membuat program dibatalkan (*system call*), atau dapat menghapus file yang ada dan membuat yang baru. Setelah dua file input dan output telah ditetapkan, maka program akan melooping membaca file input dan menulis ke file output sampai akhir file. Jika proses sudah selesai, program akan menutup kedua file dan akan terdapat pesan di layar bahwa proses telah selesai dan mengakhiri program dengan normal.

C. Multithread

Thread adalah unit dasar dari pemanfaatan CPU. Sebuah *thread* dalam proses akan menjalankan satu tugas (*job*) pada satu waktu. pada proses tradisional, setiap proses hanya memilki satu *thread*, sehingga program hanya dapat menjalankan satu tugas dalam satu waktu. namun jika proses

memilki beberapa *thread*, maka prosesnya dapat menjalakan kan beberapa tugas pada satu waktu dan hal ini disebut *multithreading*.



Gambar 11 Proses single thread dan multithread

Contohnya, jika suatu program terdiri dari proses yang memilki satu *thread* dapat memperbolehkan pengguna untuk pengetikan tulisan (*word processing*) dan pemeriksaan ejaan (*spellchecker*). Ketika pengguna sedang mengetik tulisan pada program maka fitur pemeriksaan ejaan akan berjalan setelah pengguna selesai menulis. lalu pengguna dapat menulis kembali tulisan jika *thread* telah selesai menjalankan komputasi algoritma pengejaannya. Lalu jika program tersebut memliki proses dengan dua *thread*, maka fitur pengejaan dapat berjalan secara bersamaan ketika pengguna sedang menulis tulisan pada program. Dengan ini setiap tugas tidak harus bergantian dalam menunggu tugas lainnya.

Pada prosesor dengan *singlecore* mengeksekusi proses yang memilki *thread* lebih dari satu, maka CPU akan mengeksekusi *thread-thread* tersebut dan akan bergantian secara bersisipan dalam mengeksesusi setiap threadnya. Seakan-akan sistem terdapat processing unit yang banyak namun kenyataannya processing unit bergantian menjalankan tugas secara cepat

tanpa menyelesaikan masing-masing tugas. Hal ini disebut dengan *concurrency*.

Upaya dalam membantu proses yang memliki *thread* yang banyak, mulailah berkembang teknologi baru yaitu *multicoreprocessor*. Prosesor ini adalah dimana satu *chip* prosesor memliki prosesing unit yang lebih dari satu. Sehingga sistem operasi ini melihat sistem terdapat lebih dari satu processing unit dan dapat memanfaatkan prosesor tambahan tersebut dalam menjalankan program dan proses. teknologi ini dibuat karena permintaan tenaga proses yang lebih tinggi pada waktunya. *Multicore* sistem ini dapat menjalankan *concurrency* lebih efektif dan efisien dalam menjalankan proses. peran *multicore* dalam proses yang memilki *thread* yang banyak adalah membagi rata *thread* di proses kepada masing-masing processing unit dengan rata. Contohnya jika suatu proses memilki 6 *thread* dalam eksekusinya dan dieksekusi dengan 2 *core* prosesor, maka setiap corenya dalam processornya akan mengeksekusi 3 *thread* secara *concurrent*. Proses pembagian *thread* dalam proses kepada beberapa prosesing unit disebut dengan *parallelism*, mengekseksi beberapa tugas secara bersamaan.

Multithreading adalah teknik pemrograman yang memungkinkan beberapa sub proses dalam program dapat berjalan secara paralel. Sebuah *thread* adalah sebuah bagian program yang dapat berjalan mandiri, sehingga dua atau lebih *thread* dapat berjalan bersamaan, tanpa yang satu harus menunggu selesainya yang lain. Akibatnya penggunaan *multithread* membuat GUI lebih responsip dan menggunakan *resources* menjadi lebih efektif. Keuntungan menggunakan *multithreading* diantaranya adalah responsif, berbagi sumber daya, utilisasi arsitektur multiprosesor, dan ekonomis.

Multithreading merupakan sebuah konsep untuk dapat menjalankan *task* atau tugas lebih dari satu secara paralel, sehingga dengan konsep ini *task* yang banyak akan cepat selesai karena tidak saling tunggu untuk menyelesaikan *task*. *Multithreading* adalah cara komputer untuk membagi-bagi pekerjaan yang dikerjakan sebagian-sebagian dengan cepat sehingga menimbulkan efek seperti menjalakan beberapa *task* secara bersamaan walaupun otaknya hanya satu.

Multithreading memiliki cara pengeksekusian yang mengizinkan beberapa *thread* terjadi dalam sebuah proses, saling berbagi sumber daya tetapi dapat dijalankan secara independen. Keuntungan dari sistem yang menerapkan *multithreading* dapat kategorikan menjadi 4 bagian:

1. *Responsif*

   Program dengan kemampuan untuk *multithread* dapat memperbolehkan program untuk melakukan beberapa tugas secara bersamaan atau disebut *multitasking*. Hal ini sangat memperbolehkan pengguna dapat menggunakan fungsi utama program dan secara bersamaan program dapat memberikan fitur tambahan yang membantu pengguna dalam melesaikan tugasnya tanpa menunggu pengguna berhenti menggunakn program. Hal ini yang disebut responsive yang dimana penggua dapat menerima informasi secara instan dari hasil inputnya. Contohnya pada dimana suatu website memiliki kegunaan untuk menampilkan suatu teks pdf dan dalam website tersebut terdapat tombol untuk me-*request* suatu gambar dari server. Ketika pengguna memencet tombol tersebut, proses akan menjalankan *thread* yang berperan dalam mengatur *request* dari server tanpa harus memberhentikan *thread* yang memperbolehkan untuk *scrolling* dalam menampilkan documen. Aplikasi interaktif menjadi tetap responsif meskipun sebagian dari program sedang diblok atau melakukan operasi lain yang panjang. Contohnya sebuah *thread* dari *web browser* dapat melayani permintaan pengguna sementara *thread* yang lain berusaha menampilkan gambar.

2. *Berbagi Sumber Daya*

   Sebuah proses yang terdiri dari banyak *thread* akan memilki *resource* yang sama. *resource* ini adalah termasuk data di *storage*, ruang memori dan proses *heap* dan *stack*. Dengan ini setiap tugas yang dijalankan oleh *thread* tidak perlu memesan *resources* tambahan dari sistem memori namun mengunakan *resource* yang telah berada dalam memori yang telah diambil oleh *thread* yang lain. saling membagi *resource* ini juga memungkinkan untuk efesiensi dalam dalam komputasi data yang sama. jika ada suatu data yang diperlukan untuk digunakan oleh beberapa tugas, maka *thread-thread* tersebut dapat mengakses data yang sama tanpa komputasi atau

mengambil dari *storage* tanpa mengambil data dari *storage*. Beberapa *thread* yang melakukan proses yang sama akan berbagi sumber daya. Keuntungannya adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa *thread* yang berbeda dalam lokasi memori yang sama.

3. *Ekonomis*
   Mengalokasikan memori dan *resource* lainnya untuk membuat proses sangatlah mahal. dengan itu, membuat proses baru sangat tidak efisien. Sebaliknya, membuat *thread* lebih murah dan lebih cepat. Sebuah *thread* juga lebih mudah untuk berpindah antara proses, dengan itu lebih efisien untuk saling beralih *thread* antara tugas dibandingakn untuk membuat proses baru untuk membuat tugas. Pembuatan sebuah proses memerlukan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan menggunakan *thread*, karena *thread* membagi memori dan sumber daya yang dimilikinya sehingga lebih ekonomis untuk membuat thread dan *context switching thread.*

4. *Utilitasi Arsitektur Multiprosesor*
   Keuntungan dari *multithreading* dapat sangat meningkat pada arsitektur multiprosesor, dimana setiap *thread* dapat berjalan secara paralel di atas procesor yang berbeda. Pada arsitektur processor tunggal, CPU menjalankan setiap *thread* secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi paralel, tetapi pada kenyataanya hanya satu *thread* yang dijalankan CPU pada satu-satuan waktu.
   Multithreading adalah eksekusi beberapa *thread* dari satu proses tunggal bersamaan dalam konteks proses itu. *Thread* dari suatu proses berarti segmen kode dari suatu proses, yang memiliki ID utasnya sendiri, program *counter*, register dan *stack* dan dapat dieksekusi secara independen. Tetapi *thread* yang berasal dari proses yang sama harus berbagi barang dari proses itu seperti kode, data, dan sumber daya sistem. Membuat proses terpisah untuk setiap permintaan layanan menghabiskan waktu dan sumber daya sistem pembuangan.

Perbedaan kunci antara Multi pemrosesan dan *Multithreading*:

1. Perbedaan utama; multiprosesor memungkinkan suatu sistem untuk memiliki lebih dari dua CPU yang ditambahkan ke sistem sedangkan *multithreading* memungkinkan suatu proses menghasilkan beberapa *thread* untuk meningkatkan kecepatan komputasi suatu sistem.
2. Sistem multiprosesing mengeksekusi banyak proses secara bersamaan, sedangkan sistem *multithreading* membiarkan menjalankan beberapa thread proses secara bersamaan.
3. Menciptakan suatu proses dapat menghabiskan waktu dan bahkan menghabiskan sumber daya sistem. Namun membuat *thread* adalah ekonomis karena *thread* yang berasal dari proses yang sama berbagi barang dari proses itu.
4. Multiprocessing dapat diklasifikasikan menjadi multiprocessing simetris dan multiprocessing asimetris sedangkan multithreading tidak diklasifikasikan lebih lanjut.

Multithreading meningkatkan daya tanggap seolah-olah satu utas proses diblokir atau melakukan operasi yang panjang, proses masih berlanjut. Manfaat kedua *multithreading* adalah berbagi sumber daya karena beberapa *thread* proses berbagi kode dan data yang sama dalam ruang alamat yang sama.
Membuat *thread* adalah ekonomis karena membagikan kode dan data dari proses yang mereka miliki. Jadi sistem tidak harus mengalokasikan sumber daya secara terpisah untuk setiap *thread*. *Multithreading* dapat ditingkatkan pada sistem operasi multiprosesing. Karena *multithreading* pada banyak CPU meningkatkan paralelisme.
Terdapat dua jenis *thread* yaitu:

1. *Thread* Pengguna
   Thread yang pengaturannya dilakukan oleh pustaka *thread* pada tingkatan pengguna. Karena pustaka yang menyediakan fasilitas untuk pembuatan dan penjadwalan *thread*, thread pengguna cepat dibuat dan dikendalikan.
2. *Thread* Kernel
   Thread yang didukung langsung oleh kernel. Pembuatan, penjadwalan dan manajemen *thread* dilakukan oleh kernel pada kernel *space*. Karena dilakukan oleh sistem operasi,

proses pembuatannya akan lebih lambat jika dibandingkan dengan *thread* pengguna.

Model – model *multithreading*:

1. Model *Many-to-One*
   Model ini memetakan beberapa *thread* tingkatan pengguna ke sebuah *thread* tingkatan kernel. Pengaturan *thread* dilakukan dalam ruang pengguna sehingga efisien. Hanya satu *thread* pengguna yang dapat mengakses *thread* kernel pada satu saat. Jadi *Multiple thread* tidak dapat berjalan secara paralel pada multiprosesor. Contoh: *Solaris Green Threads* dan GNU *Portable Threads*.
2. Model *One-to-One*
   Model ini memetakan setiap *thread* tingkatan pengguna ke setiap *thread*. Ia menyediakan lebih banyak *concurrency* dibandingkan model *many-to-One*. Keuntungannya sama dengan keuntungan *thread* kernel. Kelemahan model setiap pembuatan *thread* pengguna memerlukan tambahan *thread* kernel. Karena itu, jika mengimplementasikan sistem ini maka akan menurunkan kinerja dari sebuah aplikasi sehingga biasanya jumlah *thread* dibatasi dalam sistem.
3. *Model Many-to-Many*
   Model ini memultipleks banyak *thread* tingkatan pengguna ke *thread* kernel yang jumlahnya sedikit atau sama dengan tingkatan pengguna. Model ini mengizinkan developer membuat *thread* sebanyak yang ia mau tetapi *concurrency* tidak dapat diperoleh karena hanya satu *thread* yang dapat dijadwalkan oleh kernel pada suatu waktu. Keuntungan dari sistem ini ialah kernel *thread* yang bersangkutan dapat berjalan secara paralel pada multiprosessor.

D. RINGKASAN

Proses adalah program yang sedang dieksekusi. Saat proses dijalankan mengubah status. suatu proses ditentukan oleh aktivitas proses itu saat ini. Setiap proses mungkin berada di salah satu status berikut: *new*, *ready*, *running*, *waiting*; atau dihentikan. Setiap proses diwakili dalam sistem operasi oleh *Process Control Block* (PCB). Sebuah proses saat tidak dijalankan ditempatkan dalam beberapa antrian tunggu. Setiap proses diwakili oleh PCB dan PCB dapat dihubungkan bersama untuk membentuk antrian siap pakai. Biasanya, penjadwalan jangka panjang sangat dipengaruhi oleh sumber daya pertimbangan alokasi, terutama manajemen memori. Penjadwalan Jangka pendek adalah pemilihan satu proses dari antrian siap. Sistem operasi harus menyediakan mekanisme untuk proses induk untuk membuat proses. Proses yang dijalankan dalam sistem operasi dapat bersifat independen proses atau proses kooperatif. Proses kerjasama membutuhkan interpro mekanisme komunikasi untuk berkomunikasi satu sama lain. terutama, komunikasi dicapai melalui dua skema. Komunikasi dalam sistem *client-server* dapat menggunakan (1) soket, (2) prosedur panggilan jarak jauh (RPC), atau (3) pipa. Soket didefinisikan sebagai titik akhir untuk komunikasi. Koneksi antara sepasang aplikasi terdiri dari sepasang soket, satu di setiap ujung chamcel komunikasi. RPC adalah bentuk lain dari komunikasi terdistribusi. Sebuah RPC terjadi ketika sebuah proses (atau utas) memanggil prosedur pada aplikasi jarak jauh. Pipa biasa memungkinkan komunikasi antara proses induk dan anak, sementara pipa bernama mengizinkan proses yang tidak terkait untuk berkomunikasi satu sama lain

E. SOAL DAN JAWABAN

Soal:

1. Apa yang dimaksud *Process Control Block* (PCB), dan apa perbedaannya dengan *Thread*?
2. Apa itu *Batch* dan seperti apakah proses *System Calls*?
3. Apa perbedaan *user* level *thread* dan kernel *thread*?

Jawaban:

1. *Process Control Block* adalah bentuk informasi-informasi lain yang diperlukan sistem operasi untuk mengendalikan dan mengoordinasikan beragam proses aktif dalam suatu proses sedangkan *Thread* adalah unit dasar dari utilitas CPU, di dalamnya terdapat ID *thread*, program counter, register, stack dan saling berbagi dengan thread lain dalam proses yang sama.
2. *Batch* merupakan nama yang diberikan untuk tugas melakukan pekerjaan yang sama berulang-ulang, satu-satunya perbedaan yang input data disajikan untuk setiap iterasi dari pekerjaan dan mungkin file output sedangkan *System Calls* adalah penyedia antarmuka dari pelayanan-palayanan yang tersedia dengan Sistem Operasi.
3. *User thread* adalah *thread* yang mejalankan proses milih pengguna, dan kernel *thread* adalah *thread* yang di atur langsung oleh sistem operasi. Dengan itu terdapat berbagai macam relasi antara *thread space* tersebut. Model ini dimana beberapa *user thread* memilki relasi dengan satu kerel thread.

BAB 4
PENJADWALAN CPU

Penjadwalan CPU adalah dasar dari sistem operasi multiprogram. Sistem operasi dapat membuat komputer lebih produktif. Dalam bab ini, memperkenalkan penjadwalan CPU, dasar konsep dan menyajikan beberapa algoritma penjadwalan CPU serta mempertimbangkan masalah pemilihan algoritma untuk sistem tertentu.

Capaian:

1. Untuk memperkenalkan Penjadwalan CPU berbasis sistem operasi multiprogram
2. Untuk mendefiniiskan algoritma Penjadwalan CPU

A. Pengertian Penjadwalan

Penjadwalan CPU merupakan fungsi dasar dari suatu sistem operasi mengatur atau menjadwalkan proses-proses yang ada di dalam sistem komputer. Di mana masing-masing proses tersebut berjalan dan dieksekusi dalam didalam sebuah pola yang disebut *Siklus Burst*. Penjadwalan sangat penting dilakukan dalam sistem komputer dikarenakan dalam menentukan performa sebuah komputer dengan cara mengatur alokasi *resource* dari CPU untuk menjalankan proses-proses di dalam komputer. Penjadwalan CPU secara garis besar dibagi menjadi 2, yaitu Penjadwalan *Preemptive* dan Penjadwalan *Non-Preemptive*.

Keberhasilan dari penjadwalan CPU tergantung dari beberapa properti prosesor. Pengeksekusian dari proses tersebut terdiri atas siklus CPU eksekusi dan M/K *wait*. Proses hanya akan bolak balik dari dua state ini, inilah yang disebut siklus *burst* CPU-M/K. pengeksekusian proses dimulai dengan *burst* CPU, setelah itu diikuti oleh *burst* M/K, kemudian *burst* CPU lagi lalu Burst M/K lgi, dan seterusnya dilakukan bergiliran. *Burst* CPU terakhir akan berakhir dengan permintaan sistem untuk mengakhiri pengeksekusian melalui *burst* M/K lagi.

1 *Penjadwalan CPU Non-Preemptive*

Penjadwalan Non-Preemptive merupakan salah satu jenis penjadwalan di mana sistem operasi tidak pernah melakukan *context switch* dari proses yang sedang berjalan ke proses yang lain. Dengan kata lain, proses yang sedang berjalan tidak dapat di *interrupt*. Penjadwalan *non-preemptive* terjadi ketika proses hanya:

1. Berjalan dari running state sampai *waiting* state.
2. Dihentikan.

Metode ini digunakan oleh *microsoft windows* 3.1 dan *Macintosh*. Ini adalah metode yang dapat digunakan untuk meng-*interupt* pada metode penjadwalan *Preemptive*.

2 *Penjadwalan CPU Preemptive*

Penjadwalan *preemptive* mempunyai arti kemampuan sistem operasi untuk memberhentikan sementara proses yang sedang berjalan untuk memberi ruang kepada proses yang prioritasnya lebih tinggi. Penjadwalan ini dapat saja termasuk penjadwalan proses atau M/K. penjadwalan *preemptive* memungkinkan sistem untuk lebih dapat menjamin bahwa setiap proses mendapat sebuah *slice* waktu operasi. Dan juga membuat sistem lebih cepat merespon terhadap *event* dari luar (contohnya seperti ada data yang masuk) yang membutuhkan reaksi cepat dari satu atau beberapa proses. Membuat penjadwalan yang *preemptive* mempunyai keuntungan yaitu sistem lebih responsif daripada sistem yang memakai penjadwalan *non preemptive*. Dalam waktu-waktu tertentu, proses dapat dikelompokkan ke dalam dua kategori: proses yang memiliki *burst* M/K yang sangat lama disebut I/O *bound*, dan proses yang memiliki *burst* CPU yang sangat lama disebut CPU *bound*. Terkadang juga suatu sistem mengalami kondisi yang disebut *busy wait*, yaitu saat di mana sistem menunggu *request* input (seperti disk, *keyboard*, atau jaringan). Saat *busy wait* tersebut, proses tidak melakukan sesuatu yang produktif, tetapi tetap memakan *resource* dari CPU. Dengan penjadwalan *preemptive*, hal tersebut dapat dihindari. Dengan kata lain, penjadwalan *preemptive* melibatkab mekanisme interupsi yang menyela proses yang

sedang berjalan dan memaksa sistem untuk menentukan proses mana yang akan dieksekusi selanjutnya. Lama waktu suatu proses diizinkan untuk dieksekusi dalam penjadwalan *preemptive* disebut *time slice*/*quantum*.

Penjadwalan berjalan setiap satu satuan *time slice* untuk memilih proses mana yang akan berjalan selanjutnya. Bila *time slice* terlalu pendek maka penjadwal akan memakan terlalu banyak waktu proses, tetapi bila *time slice* terlalu lama maka memungkinkan proses untuk tidak dapat merespon terhadap *event* dari luar secepat yang diharapkan.

Algoritma penjadwalan CPU yang berbeda memiliki properti yang berbeda, dan pilihannya dari algoritma tertentu dapat mendukung satu kelas proses di atas yang lain. Algoritma mana yang akan digunakan dalam situasi tertentu, harus mempertimbangkan properti dari berbagai algoritma. banyak kriteria telah disarankan untuk membandingkan algoritma penjadwalan CPU. Karakteristik yang digunakan untuk perbandingan dapat membuat perbedaan di mana algoritma dinilai sebagai yang terbaik.

B. Algoritma Penjadwalan

Penjadwalan CPU berkaitan dengan masalah memutuskan proses mana dalam antrian siap akan dialokasikan CPU. Ada banyak penjadwalan CPU yang berbeda algoritma. Berikut ini jenis algortima penjadwalan adalah:

1 First Come, First Served Schedulling

Sejauh ini, algoritma penjadwalan CPU yang paling sederhana adalah yang pertama datang, yang pertama dilayani (FCFS) algoritma penjadwalan. Dengan skema ini, proses yang meminta CPU terlebih dahulu dialokasikan CPU terlebih dahulu. Implementasi kebijakan FCFS adalah mudah dikelola dengan antrian FIFO. Ketika sebuah proses memasuki antrian siap, PCB terhubung ke ekor antrian. Proses yang berjalan kemudian dihapus dari antrian. Kode untuk penjadwalan FCFS mudah untuk ditulis dan dipahami. Penjadwalan ini terdiri dari prosess dan *Burst Time*

Process	Burst Time
P_1	24
P_2	3
P_3	3

Gambar 12 Gantt chart first come

Waktu tunggu adalah 0 milidetik untuk proses P1, 24 milidetik untuk proses P2, dan 27 milidetik untuk proses P3. Jadi, waktu tunggu rata-rata adalah (0 + 24 + 27)/3 = 17 milidetik. Jika proses tiba dalam urutan P2, P3, P1, namun, hasilnya akan seperti yang ditunjukkan pada bagan *gantt* berikut:

Waktu tunggu rata-rata sekarang (6 + 0 + 3)/3 = 3 milidetik. Pengurangan ini substansial. Dengan demikian, waktu tunggu rata-rata di bawah kebijakan FCFS umumnya tidak minimal dan dapat bervariasi secara substansial.

2 Shortest Job First

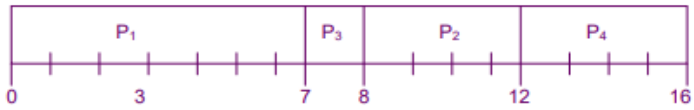
Di dalam penjadwalan *shortest job first* (SJF) yang dilayani dahulu adalah proses yang terdapat CPU *burst* terkecil. Ada dua skema berikut ini:

1. *Non-Preemptive*, apabila CPU diberikan pada proses, maka tidak dapat ditunda sampai CPU burst selesai.
2. *Preemptive*, jika proses baru datang dengan panjang CPU burst lebih pendek dari sisa waktu proses yang saat itu sedang dieksekusi, proses ini ditunda dan diganti dengan proses baru. Skema ini disebut dengan Shortest-Remaining-Time-First (SRTF).

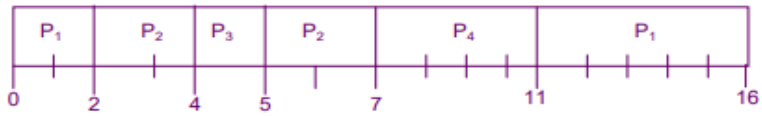
SJF adalah algoritma penjadwalan yang optimal dengan rata-rata waktu tunggu yang minimal. Misalnya terdapat empat proses dengan panjang CPU burst dalam milidetik.

Prosess	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Penjadwalan proses dengan algoritma SJF (*non preemptive*) dapat dilihat pada *gant chart* berikut:



Waktu tunggu untuk P_1 adalah 0, P_2 adalah 26, P_3 adalah 3 dan P_4 adalah 7 sehingga rata-rata waktu tunggu adalah (0 + 6 + 3 + 7)/4 = 4 milidetik. Sedangkan Penjadwalan proses dengan algoritma SRTF (*preemptive*) dapat dilihat pada *gant chart* berikut:



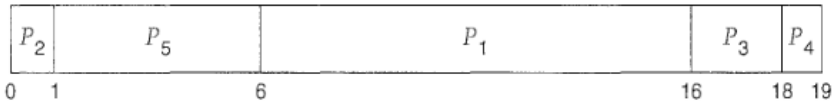
Waktu tunggu untuk P_1 adalah 9, P_2 adalah 1, P_3 adalah 0 dan P_4 adalah 4 sehingga rata-rata waktu tunggu adalah (9 + 1 + 0 + 4)/4 = 3 milidetik.

3 *Priority Schedulling*

Algoritma SJF adalah kasus khusus dari algoritma penjadwalan prioritas umum. Prioritas dikaitkan dengan setiap proses, dan CPU dialokasikan ke proses dengan prioritas tertinggi. Proses dengan prioritas yang sama dijadwalkan dalam urutan FCFS. Sebuah algoritma SJF hanyalah sebuah algoritma prioritas di mana prioritas (p) adalah kebalikan dari

(diprediksi) CPU *burst* berikutnya. Semakin besar CPU *burst*, semakin rendah prioritas, dan sebaliknya. Sebagai contoh terdapat 5 proses yaitu:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



Prioritas dapat ditentukan baik secara internal maupun eksternal. Didefinisikan secara internal prioritas menggunakan beberapa kuantitas atau kuantitas yang dapat diukur untuk menghitung prioritas dari sebuah proses. Misalnya, batas waktu, kebutuhan memori, jumlah membuka file, dan rasio rata-rata I/0 terhadap CPU rata-rata telah digunakan dalam komputasi prioritas. Prioritas eksternal ditetapkan oleh kriteria di luar sistem operasi, seperti pentingnya proses, jenis dan jumlah dana yang dibayarkan untuk penggunaan komputer, departemen yang mensponsori pekerjaan, dan lainnya, seringkali faktor politik. Penjadwalan prioritas dapat berupa preemptive atau nonpreemptive. Ketika sebuah proses tiba di antrian siap, prioritasnya dibandingkan dengan prioritas dari proses yang sedang berjalan. Sebuah algoritma penjadwalan prioritas preemptive akan mendahului CPU jika prioritas proses yang baru tiba lebih tinggi daripada prioritas proses yang sedang berjalan.

4 Round Robin Schedulling

Algoritma penjadwalan *round-robin* (RR) dirancang khusus untuk sistem berbagi. Ini mirip dengan penjadwalan FCFS, tetapi *preemption* ditambahkan ke memungkinkan sistem untuk beralih di antara proses. Satuan waktu kecil yang disebut kuantum waktu atau irisan waktu, didefinisikan. Sebuah kuantum waktu umumnya dari 10nhingga 100 milidetik. Antrian siap diperlakukan sebagai antrian melingkar.

Penjadwal CPU berjalan di se r antrian siap, mengalokasikan CPU ke masing-masing proses untuk interval waktu hingga 1 kali kuantum.

C. RINGKASAN

Penjadwalan CPU adalah tugas memilih proses menunggu dari antrian siap dan mengalokasikan CPU untuk itu. CPU dialokasikan ke proses yang dipilih oleh petugas operator Penjadwalan *first-come, first-served* (FCFS) adalah algoritma penjadwalan yang paling sederhana, tetapi dapat menyebabkan proses yang singkat menunggu proses yang sangat lama. Penjadwalan (SJF) terbukti optimal, memberikan rata-rata terpendek waktu menunggu. Menerapkan penjadwalan SJF sulit. Algoritma SJF adalah algoritma penjadwalan prioritas umum, yang hanya mengalokasikan CPU ke proses dengan prioritas tertinggi. Penjadwalan *round-robin* (RR) lebih sesuai untuk *time-shared* (interaktif) sistem. Penjadwalan RR mengalokasikan CPU ke proses pertama yang siap antrian untuk q unit waktu, di mana q adalah kuantum waktu. Setelah q satuan waktu, jika prosesnya belum melepaskan CPU, prosesnya sudah di *preemption*, dan prosesnya di letakkan pada ujung antrian *ready*. Masalah utama adalah pemilihan kuantum waktu. Jika kuantum terlalu besar, penjadwalan RR merosot ke FCFS penjadwalan; jika kuantum terlalu kecil, penjadwalan *overhead* dalam bentuk waktu peralihan konteks menjadi berlebihan. Algoritma FCFS adalah *nonpreemptive*; algoritma RR bersifat *preemptive*. SJF dan algoritma prioritas dapat berupa *preemptive* atau *nonpreemptive*

D. SOAL DAN JAWABAN

Soal:

1. Jelaskan pengertian dari algoritma penjadwalan *shortest job first* (SJF)!

Jawab:
Algoritma Shortest Job First (SJF) adalah salah satu algoritma penjadwalan dimana proses yang akan didahulukan pengerjaannya adalah proses yang memiliki waktu proses terpendek. Hal ini mengakibatkan setiap proses dalam antrian memiliki waktu tunggu (waiting time) yang pendek

BAB 5
SINKRONISASI

Sinkronisasi, adalah proses pengaturan jalannya beberapa proses pada saat yang bersamaan. Tujuan utama sinkronisasi adalah menghindari terjadinya inkonsitensi data karena pengaksesan oleh beberapa proses yang berbeda (*mutual exclusion*) serta untuk mengatur urutan jalannya proses-proses sehingga dapat berjalan dengan lancar dan terhindar dari *race condition*, *deadlock* dan *starvation*

Capaian:

1. Menjelaskan tentang sinkronisai
2. Menjelaskan tentang Algoritma dalam sinkronisasi

Sinkronisasi dapat diartikan juga sebagai komunikasi antara proses yang membutuhkan *place by calls* untuk mengirim dan menerima data *primitive*. Terdapat rancangan yang berbeda-beda dalam implementasi setiap *primitive*. Pengiriman pesan mungkin dapat diblok atau tidak dapat dibloking juga dikenal dengan nama sinkron atau asinkron. Pengertian lain, sinkronisasi adalah proses pengaturan jalannya beberapa proses pada saat yang bersamaan. Sinkronisasi umumnya dilakukan dengan bantuan perangkat sinkronisasi. Penyelesaian terhadap masalah ini sangat penting karena perkembangan teknologi sistem komputer menuju ke sistem *multiprocessing*, terdistribusi dan paralel yang mengharuskan adanya proses-proses konkuren.

Konkurensi adalah kondisi dimana pada saat yang bersamaan terdapat lebih dari satu proses disebut dengan konkurensi (proses-proses yang kongkuren). Proses-proses yang mengalami kongkuren dapat berdiri sendiri atau dapat saling berinteraksi, sehingga membutuhkan sinkronisasi atau koordinasi proses yang baik. Untuk penanganan konkuren, bahasa pemrograman saat ini telah memiliki mekanisme kongkurensi dimana dalam penerapannya perlu dukungan sistem operasi dimana bahasa berada. Sinkronisasi diperlukan

untuk menghindari terjadinya ketidak konsistenan data akibat adanya akses data secara konkuren. Proses-proses disebut konkuren jika proses-proses itu ada dan berjalan pada waktu yang sama, proses-proses konkuren ini dapat bersifat *independent* atau dapat juga saling berinteraksi. Proses-proses konkuren yang saling berinteraksi memerlukan sinkronisasi agar terkendali dan juga menghasilkan output yang benar.

A. Komunikasi antar Proses

Salah satu kelemahan monitor saat ini dan data mengontrol saling pengecualian adalah bahwa semua *thread* atau prosesor yang bersamaan memiliki akses ke memori bersama dan berbagi kernel sistem operasi yang sama (yang bertanggung jawab untuk membuat proses tertidur dan membangunkannya). Hal ini banyak berlaku pada pesan dan jarang berlaku pada mesin primitif. Mereka memungkinkan untuk menyinkronkan proses (dengan menunggu pesan) dan menukar data antar proses. Selain itu, mereka dapat bekerja di berbagi *thread* dan proses pada mesin yang sama serta berbeda

1. Message Passing

a. Bentuk komunikasi antar proses yang menggunakan dua primitif
   - *Send* (*destination* dan *message*): Mengirim pesan ke tujuan tertentu.
   - *Receive* (*source* and *message*): Menerima pesan dari sumber. Panggilan ini dapat memblokir jika tidak ada pesan.

b. Keuntungan Pesan:
   - Konsep pesan dapat disalahkan dari lingkungan prosesor tunggal ke sistem jaringan dari beberapa prosesor masing-masing dengan memori sendiri.

c. Kekurangan Pesan:

- Pada beberapa prosesor ada kemungkinan penyampain pesan lebih lambat dan *overhead* penyalinan pesan. Namun beberapa sistem membuat penyimpanan pesan lokal menjadi efisien.

2. *Messages Rendezvou*

   Rendozvius adalah varian penyampaian pesan yang tidak menggunakan *buffering* pesan. Seorang pengirim memblokir sampai penerima membaca pesan tersebut. Jika pengiriman sampai terjadi penerimaan. Dalam lingungan SMP (*Multiprosesor simetris*), sebagai lawan dari lingkungan multi komputer, pesan dapat disalin secara langsung tanpa *buffering* perantara. Jika penerimaan dilakukan terlebih dahulu, ia memblokir hingga pengiriman terjadi.

   - Keuntungan redzvous
     Mudah dan efisien untuk diterapkan.
   - Kerugian rendzvous
     Memaksa pengiriman dan Penerimaan untuk disinkronisasi dengan erat pada persimpangan kirim/terima.

3. *Mailbox*

   Pesan dan pertemuan membutuhkan pengalaman langsung di mana proses pengiriman harus mengidentifikasi proses penerimaan. Proses peneriman dapat menjadi parameter yang masuk opsi penerimaan. Mekanisme ini dapat memudahkan dalam satu pengiriman/penerima tunggal atau beberapa model pengiriman. Proses pengiriman ini memiliki beberapa cara untuk menerimanya.

   Mailbox adalah entitas perantara yang diterapkan pada perpesanan. *Mailbox* ini memudahkan banyak pembaca karena mereka semua dapat menghubungi *Mailbox* yang sama dan mengekstrak item dan antrian. Dan jika kotak surat penuh maka proses pengiriman ditunda atau dihapus.

   - Keuntungan dari *mailbox*:

   Memberikan fleksibilitas untuk memiliki banyak pengirim atau penerima. Selain itu, tidak mengharuskan pengirim dan penerima perlu mengetahui koordinasi tempat surat.

- Kerugian dari *mailbox*:

Mereka mungkin harus dikenakan tingkat menyalin data ekstra karena data harus terlebih dahulu ke antrian kotak surat dan kemudian disalin penerima. Dalam sistem jaringan ada pertanyaan dimana kotak surat harus berada. Karena pengirim dan penerima semuanya dapat menjadi komputer yang berbeda, kotak surat menjadi tujuan yang berbeda di jaringan: Pesan dikirim ke kotak surat dan kemudian diambil dari kotak surat. Tindakan ini memberlakukan lompatan ekstra melalui jaringan untuk pesan.

B. Algoritma Sikronisasi

Algoritma untuk sinkronisasi dalam system terdistribusi memiliki beberapa sifat:

1) Informasi yang relevan tersebar di beberapa computer
2) Keputusan pembuatan proses hanya berdasarkan informasi local.
3) Peristiwa kegagalan dengan penyebab tunggal di dalam sistem harus dihindarkan
4) Tidak tersedianya clock atau sumber waktu global yang akurat.

Sikronisasi merupakan bagian penting untuk kerjasama dalam pemakaian sumberdaya berbagi (*Sharing resources*), pengurutan kejadian dan kesepakatan clock tersebar.

Clock logika

Boleh dikatakan semua komputer memiliki rangkaian pencatat waktu. Walaupun penggunakan kata *Clock* sudah meluas, kata yang lebih tepat adalah *timer* untuk merujuk komponen dari rangkaian tersebut. *Timer* ini menggunakan *crystal quartz* sebagai sumber frekuensinya. Walaupun frekuensi osilator pada osilator kristal biasanya stabil, tetap saja tidak mungkin menjamin bahwa semua kristal yang bekerja diberbagai komputer memiliki frekuensi yang persis sama.

Selalu ada sedikit perbedaan yang terjadi dan mengakibatkan perbedaan waktu pula yang disebut *clock skew*. Berbagai

algoritma telah dikembangkan untuk menangani sinkronisasi *clock* dan beberapanya akan dibahas berikut ini.

C. Algoritma Petterson

Selanjutnya akan mengilustrasikan solusi berbasis perangkat lunak klasik untuk masalah bagian kritis yang dikenal sebagai solusi Peterson. Karena cara arsitektur komputer modern melakukan instruksi bahasa mesin dasar, seperti memuat dan menyimpan, tidak ada jaminan bahwa solusi Peterson akan bekerja dengan benar pada arsitektur tersebut. Namun, algoritma petterson menyajikan solusi yang memberikan deskripsi algoritmik yang baik untuk memecahkan masalah bagian kritis dan menggambarkan beberapa kompleksitas yang terlibat dalam merancang perangkat lunak yang membahas persyaratan saling pengecualian, kemajuan, dan menunggu bom. Solusi Peterson dibatasi pada dua proses yang mengeksekusi secara bergantian antara bagian kritis dan bagian sisa. Proses diberi nomor Po dan P1. Untuk kenyamanan, saat menyajikan Pi, kami menggunakan Pj untuk menunjukkan proses lainnya; yaitu, j sama dengan 1 – i. Solusi Peterson membutuhkan dua proses untuk berbagi dua item data:

- int turn;
- boolean flag[2];

D. RINGKASAN

Sinkronisasi dapat diartikan juga sebagai komunikasi antara proses yang membutuhkan place by calls untuk mengirim dan menerima data primitive. Terdapat rancangan yang berbeda-beda dalam implementasi setiap primitive. Pengiriman pesan mungkin dapat diblok (blocking) atau tidak dapat dibloking (nonblocking) juga dikenal dengan nam a sinkron atau asinkron. Pengertian lain, sinkronisasi adalah proses pengaturan jalannya beberapa proses pada saat yang bersamaan. Sinkronisasi umumnya dilakukan dengan bantuan perangkat sinkronisasi. Penyelesaian terhadap masalah ini sangat penting karena perkembangan teknologi sistem komputer menuju ke sistem

multiprocessing, terdistribusi dan paralel yang mengharuskan adanya proses-proses konkuren.

E. SOAL DAN JAWABAN

Soal:

2. Apa yang dimaksud dengan sinkronisasi?
3. Sebutkan Kekurangn dan kelenihan dari Massage Passing?
4. Apa yang di maksud dengan starvation?

Jawab:

1. Sinkronisasi adalah komunikasi antara proses yang membutuhkan place by calls untuk mengirim dan menerima data primitive.
2. Keuntungan pesan:
   Konsep pesan dapat disalahkan dari lingkungan prosesor tunggal ke sistem jaringan daari beberapa prosesor, masing-masing dengan memori sendiri
   Kekurangan Pesan:
   Bahkan pada beberapa prosesor, ada kemungkinan penyampain pesan lebih lambat dan overhead penyalinan pesan. Namun beberapa sistem membuat penyimpanan pesan lokal menjadi efisien.
3. Starvision merupakan keadaan dimana pemberian akses bergantian terus-menerus, dan ada suatu proses yang tidak mendapatkan gilirannya.

BAB 6
DEADLOCK

Deadlock adalah suatu kondisi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan *resource* yang sedang dipakai. Karena beberapa proses itu saling menunggu, maka tidak terjadi kemajuan dalam kerja proses-proses tersebut.

Capaian:

1. Mengetahui Pengertian *Deadlock*
2. Menjelaskan tentang Solusi *Deadlock*

Deadlock juga mengandung pengertian sebagai suatu kondisi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan resource yang sedang dipakai. Karena beberapa proses itu saling menunggu, maka tidak terjadi kemajuan dalam kerja proses-proses tersebut. *Deadlock* dalam arti sebenarnya adalah kebuntuan. Kebuntuan yang dimaksud dalam sistem operasi adalah kebuntuan proses. *Deadlock* adalah masalah yang biasa terjadi ketika banyak proses yang membagi sebuah *resource* yang hanya boleh dirubah oleh satu proses saja dalam satu waktu. Jadi *deadlock* ialah suatu kondisi dimana proses tidak berjalan lagi atau pun tidak ada komunikasi lagi antar proses. Salah satu contoh sebuah *deadlock*:

1. Proses A menggunakan CD-ROM.
2. Proses B menggunakan Scanner.
3. Proses A meminta Scanner (tanpa melepas CD-ROM) → menunggu.
4. Proses B meminta CD-ROM (tanpa melepas Scanner) → menunggu.
5. Menunggu… menunggu… DEADLOCK

A. Model Sistem Deadlock

Model sistem *deadlock* dimodelkan dengan menggunakan sekumpulan sistem. Untuk memodelkan kondisi *deadlock*, maka bayangkan sebuah sistem dengan:

1. Sekumpulan proses, P = {P1, P2, …, Pn}
2. Sekumpulan tipe sumber daya yang berbeda, R = {R1, R2, …, Rm}
3. Sumber daya Ri memiliki n bagian (*instans*) yang identik dan masing-masing digunakan.

Pada model operasi normal, sebuah proses menggunakan sumber daya dengan urutan sebagai berikut:

- Mengajukan permohonan (*request*)
  Bila permohonan tidak dapat dikabulkan dengan segera (misal karena sumber daya sedang digunakan proses lain), maka proses itu harus menunggu sampai sumber daya yang dimintanya tersedia.
- Menggunakan sumber daya (*use*)
  Proses dapat menggunakan sumber daya, misalnya printer untuk mencetak, disk drive untuk melakukan operasi I/O, dan sebagainya.
- Melepaskan sumber daya (*relase*)
  Setelah proses menyelesaikan penggunaan sumber daya, maka sumber daya harus dilepaskan sehingga dapat digunakan oleh proses lain.

B. Resource Deadlock

Deadlock dapat terjadi pada saat proses akan mengakses obyek secara tidak semestinya. Obyek tersebut dinamakan sumber daya. Sumber daya ada dua jenis, yaitu:

1. *Preemptable*
   Sumber daya dikatakan *preemptable* jika sumber daya tersebut dapat diambil (dilepas) dari proses yang sedang memakainya tanpa member efek apapun pada proses tersebut. Sumber daya ini tidak habis dipakai oleh proses mana pun. Tetapi setelah proses berakhir, sumber daya ini dikembalikan untuk dipakai oleh proses lain yang sebelumnya tidak kebagian sumber daya ini.
2. *Non Preemptable*

Pada sumber daya jenis ini, sumber daya tidak dapat diambil dari proses yang sedang membawanya karena akan menimbulkan kegagalan komputasi. Printer adalah salah satu contohnya. Jika suatu proses sedang menggunakan printer untuk mencetak sesuatu, maka printer tersebut tidak dapat diambil untuk mencetak sesuatu dari proses lain. Sumber daya jenis ini biasanya berpotensi terjadinya *deadlock*.

C. Penyebab *Deadlock*

Setelah melihat beberapa illustrasi di atas, mungkin sekarang mulai dapat membayangkan apa itu *deadlock*. Sebenarnya *deadlock* itu akan terjadi apabila syarat-syarat dari *deadlock* tersebut terpenuhi. Adapun 4 kondisi penyebab *deadlock* adalah sebagai berikut:

1. *Mutual Exclusion*
   Hanya ada satu proses yang boleh memakai sumber daya, dan proses lain yang ingin memakai sumber daya tersebut harus menunggu hingga sumber daya tadi dilepaskan atau tidak ada proses yang memakai sumber daya tersebut.
2. *Hold and wait*
   Proses yang sedang memakai sumber daya boleh meminta sumber daya lagi maksudnya menunggu hingga benar-benar sumber daya yang diminta tidak dipakai oleh proses lain, hal ini dapat menyebabkan kelaparan sumber daya sebab dapat saja sebuah proses tidak mendapat sumber daya dalam waktu yang lama.
3. *No Preemption*
   Sumber daya yang ada pada sebuah proses tidak boleh diambil begitu saja oleh proses lainnya. Untuk mendapatkan sumber daya tersebut, maka harus dilepaskan terlebih dahulu oleh proses yang memegangnya, selain itu seluruh proses menunggu dan mempersilahkan hanya proses yang memiliki sumber daya yang boleh berjalan.

4. *Circular Wait*
   Kondisi seperti rantai, yaitu sebuah proses membutuhkan sumber daya yang dipegang proses berikutnya.

D. Solusi *Deadlock*

Deadlock adalah masalah yang biasa terjadi ketika banyak proses yang membagi sebuah pendapat yang hanya boleh dirubah oleh satu saja dalam satu waktu. Strategi untuk menghadapi *deadlock* dapat dibagi menjadi tiga metode, yaitu *Prevention* (Pencegahan), *Avoidance* (Penghindaran), *Detection* (Pendeteksian).

1 Prevention (Pencegahan)

Prevention (Pencegahan) merupakan metode yang melakukan pencegahan terhadap 4 kondisi yang menyebabkan *deadlock*. Kondisi tersebuat yaitu:

- Mencegah *Mutual Exclution*

Mutual exclusion merupakan suatu kondisi yang benar-benar tak dapat dihindari. Hal ini dikarenakan tidak ada sumber daya yang dapat digunakan bersama-sama, jadi sistem harus membawa sumber daya yang tidak dapat digunakan bersama-sama.

- Mencegah *Hold & Wait*

Mencegah kondisi *hold and wait*, sistem harus menjamin bila suatu proses meminta sumber daya, maka proses tersebut tidak sedang memegang sumber daya yang lain. Proses harus meminta dan dialokasikan semua sumber daya yang diperlukan sebelum proses memulai eksekusi atau mengijinkan proses meminta sumber daya hanya jika proses tidak membawa sumber daya lain. Model ini mempunyai utilitas sumber daya yang rendah dan kemungkinan terjadi starvation jika proses membutuhkan sumber daya yang popular sehingga terjadi keadaan menunggu yang tidak terbatas karena setidaknya satu dari sumber daya yang dibutuhkannya dialokasikan untuk proses yang lain.

- Mencegah *Non-Preemption*

Peniadaan *non preemption* mencegah proses-proses lain harus menunggu. Seluruh proses menjadi *preemption*, sehingga tidak ada tunggu menunggu. Cara mencegah kondisi *non preemption*:

- Jika suatu proses yang membawa beberapa sumber daya meminta sumber daya lain yang tidak dapat segera dipenuhi untuk dialokasikan pada proses tersebut, maka semua sumber daya yang sedang dibawa proses tersebut harus dibebaskan.
- Proses yang sedang dalam keadaan menunggu, sumber daya yang dibawanya ditunda dan ditambahkan pada daftar sumber daya.
- Proses akan di restart hanya jika dapat memperoleh sumber daya yang lama dan sumber daya baru yang diminta.

2 Avoidance (Penghindaran)

Avoidance (Penghindaran) metode dengan digunakannya informasi tambahan tentang bagaimana sumber daya diminta.

- Proses harus menyatakan seluruh sumber daya maksimum yang dibutuhkan sebelum eksekusi.
- Ketika eksekusi berlangsung, proses meminta sumber daya yang diperlukan hingga batas maksimum yang dinyatakan di awal.
- Proses yang menyatakan kebutuhan melewati kapasitas sistem, tidak akan dieksekusi.

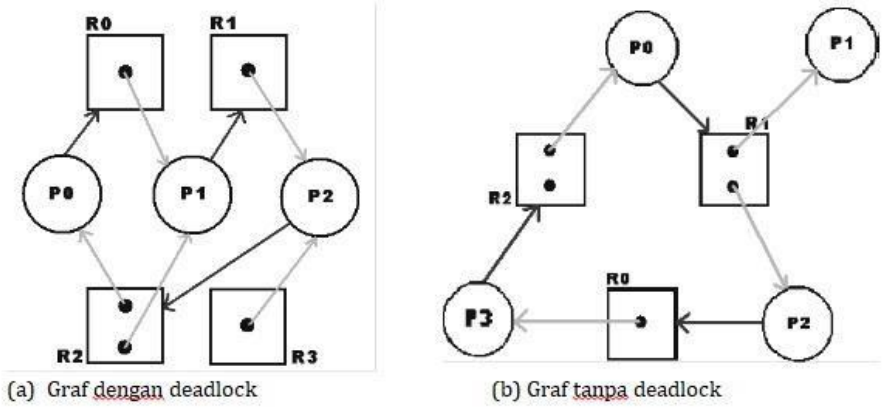
3 Safe State

State dinyatakan *safe state* jika tidak *deadlock* dan terdapat cara untuk memenuhi seluruh permintaan tanpa menghasilkan *deadlock*. Urutan proses aman jika untuk setiap Pi, sumber daya yang masih diminta Pi masih memenuhi sumber daya yang tersedia dan sumber daya yang dibawa oleh setiap Pj, dimana j < i. Jika sumber daya yang diperlukan Pi tidak dapat segera disediakan, maka Pi dapat menunggu sampai semua Pj selesai. Ketika Pj selesai, Pi dapan memperoleh sumber daya yang diperlukan, mengeksekusi, mengembalikan sumber daya yang dialokasikan dan terminasi. Ketika Pi selesai, Pi+1 dapat memperoleh sumber daya yang diperlukan dan seterusnya.

4 Detection (Pendeteksian)

Untuk mengetahui ada tidaknya deadlock dalam suatu graf dapat dilihat dari perputaran dan resource yang dimilikinya, yaitu:

- Jika tidak ada perputaran berarti tidak *deadlock*.
- Jika ada perputaran, ada potensi terjadi *deadlock*.
- *Resource* dengan instan tunggal dan perputaran mengakibatkan *deadlock*.



Gambar 13 Graf dengan dan tanpa deadlock

Pada gambar "*Graf* dengan *deadlock*", terlihat bahwa ada perputaran yang memungkinkan tejadinya *deadlock* dan semua sumber daya memiliki satu instans kecuali sumber daya R2.

Graf tersebut memiliki minimal dua perputaran, yaitu:

- R2 -> P0 -> R0 -> P1 -> R1 -> P2 -> R2
- R2 -> P1 -> R1 -> P2 -> R2

E. Algoritma Deadlock

Algoritma *Banker* dikemukakan oleh Edsger W.Dijkstra dan merupakan salah satu metode untuk menghindari *deadlock*. Algoritma penjadualan ini diungkapkan oleh Dijkstra (1965) lebih dikenal dengan nama Algoritma *Bankir*. Algoritma ini mencegah terjadinya *deadlock* dengan memutuskan apakah menyetujui atau menunda permintaan sumber daya oleh

proses. Ketika sebuah proses meminta sumber daya, maka permintaan tersebut harus diperiksa oleh *bankir*. Agar algoritma *banker* dapat dijalankan diperlukan data-data antara lain, *allocation resource*, *max requirement*, *resource need*, *available resource*, dan *total resource*.

(a) *Allocation resource* adalah jumlah alokasi *resource* yang diterima/dibawa yang dikunci oleh proses pada waktu itu. *Allocation resource* dapat dicari dengan rumus *max requirement - resource need*

(b) *Max requirement* adalah kebutuhan *maximum resource* yang diminta oleh setiap proses dan berguna agar eksekusi proses berjalan sempurna. *Max requirement* dapat dicari dengan rumus *allocation resouece+resource need*

(c) *Resource need* adalah kebutuhan *resource* selain dari *allocation resource*. *Resource* yang tetap dibutuhkan agar jumlah mencapai maksimum. *Resource need* dapat dicari menggunakan rumus *max requirement – allocation resource*

(d) *Available resource* adalah *resource* yang masih tersedia dan bebas digunakan apabila proses itu membutuhkan. *Available resource* dapat dicari dengan rumus *total resource-total(allocation resource)*.

(e) *Total resource* adalah jumlah total secara keseluruhan. *Total resource* ini dapat dicari menggunakan rumus *available resource+total(allocation resource).*

F. Algoritma Banker

(1) Langkah pertama dari algoritma banker adalah cek *Resource Need* nya

(2) Jika *Available* lebih besar atau sama dengan *Resource Need* maka *Available* dapat dipinjamkan untuk di eksekusi. Lalu mengubah nilai *Available* selanjutnya menjadi *Available* yang dipinjamkan ditambah dengan *Allocation* Proses (Urutan ditulis)

(3) Jika *Available* lebih kecil dari *Resource Need* maka tidak ada pengubahan nilai *Available* atau dilewati (Tidak ditulis nomor urutannya dan proses dilanjutkan lagi)

(4) Jika sudah melewati seleksi itu cek apakah masih ada proses yang masih belum dilaksanakan/dilewati. Jika masih ada maka

akan diproses lagi dari langkah awal, jika tidak ada maka akan selesai.

Waktu	Process	Allocation			Max. Req			Need			Available			Keterangan	Urutan Aman (safe sequence)
		A	B	C	A	B	C	A	B	C	A	B	C		
1	P0	0	1	0	7	5	3	7	4	3	3	3	2	Available utk need P0 kurang shg dilewati	
2	P1	2	0	0	3	2	2	1	2	2	3	3	2	Available utk need P1 banyak shg bisa dipinjamkan, update available	P1
3	P2	3	0	2	9	0	2	6	0	0	5	3	2	Available utk need P2 kurang shg dilewati	
4	P3	2	1	1	2	2	2	0	1	1	5	3	2	Available utk need P3 banyak shg bisa dipinjamkan, update available	P1, P3
5	P4	0	0	2	4	3	3	4	3	1	7	4	3	Available utk need P4 banyak shg bisa dipinjamkan, update available	P1, P3, P4
6	P0	0	1	0	7	5	3	7	4	3	7	4	5	Available utk need P0 banyak shg bisa dipinjamkan, update available	P1, P3, P4, P0
7	P2	3	0	2	9	0	2	6	0	0	7	5	5	Available utk need P2 banyak shg bisa dipinjamkan, update available	P1, P3, P4, P0, P2
Available Update = Total Resource											**10**	**5**	**7**	**Urutan untuk Safe Sequence**	**P1, P3, P4, P0, P2**

Gambar 14 Algoritma banker

G. Algoritma Safety

Algoritma ini adalah algoritma yang dipakai untuk menentukan apakah sebuah sistem berada dalam keadaan *safe state* atau *unsafe state*.

Berdasarkan contoh algoritma *banker* sebelumnya, diperoleh *safe sequence* berupa

P1-P3-P4-P0-P2. Sehingga algoritma *safety* mengatakan proses dengan *resource* tersebut dinyatakan di status aman (*safe state*), karena semua proses dapat dieksekusi. Algoritma *safety* akan ada di status aman apabila jumlah proses yang dieksekusi lebih dari 50%, jika dibawah syarat tersebut maka dinyatakan pada status tidak aman (*unsafe state*) **RESORUCE ALLOCARTOR DEADLOCK**

Deadlock dapat digambarkan lebih presisi dengan menggunakan *graph* berarah yang disebut *resource allocation graph*. *Graph* terdiri dari himpunan titik *V* dan garis *E*. Himpunan titik (vertex) *V* dibagi menjadi dua tipe yaitu himpunan proses yang aktif pada sistem $P = \{P_1, P_2, ..., P_n\}$ dan tipe sumber daya pada sistem $R = \{R_1, R_2, ..., R_m\}$

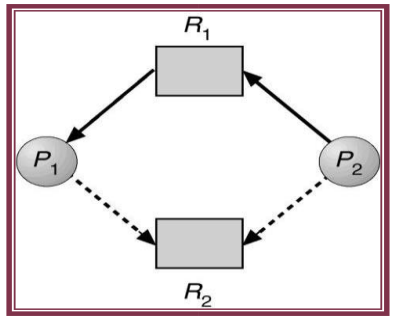
Garis berarah dari proses P_i ke tipe sumber daya R_j dinotasikan dengan $P_i \rightarrow R_j$ artinya proses P_i meminta satu anggota dari tipe sumber daya R_j dan sedang menunggu sumber daya tersebut. Garis berarah dari tipe sumber daya R_j ke proses P_i dinotasikan dengan $R_j \rightarrow P_i$ artinya satu anggota tipe sumber daya R_j dialokasikan ke proses P_i. Garis berarah

$P_i \rightarrow R_j$ disebut *request edge* dan garis berarah $R_j \rightarrow P_i$ disebut *assignment edge*.

H. Resource Allocator Graph

Untuk menghindari *deadlock* pada sistem yang hanya mempunyai satu anggota untuk setiap tipe sumber daya, dapat digunakan algoritma *resource allocation graph*. *Claim edge* $P_i \rightarrow R_j$ menandakan bahwa proses P_i mungkin meminta sumber daya R_j yang direpresentasikan dengan garis putus-putus. *Claim edge* akan berubah ke *request edge* bila proses meminta sumber daya. Sumber daya sebelumnya harus diklaim pada sistem. Sehingga sebelum proses P_i mulai dieksekusi, semua *claim edge* harus muncul pada *resource allocation graph*.

Misalnya proses Pi meminta sumber daya Rj. Permintaan dapat dipenuhi hanya jika mengubah *request edge* $P_i \rightarrow R_j$ ke *assignment edge* $R_j \rightarrow P_i$ tidak menyebabkan siklus pada graph. Jika tidak terdapat siklus, maka alokasi sumber daya menyebabkan sistem dalam state aman. Jika terjadi siklus, maka alokasi akan membawa sistem pada state tak aman. Sehingga proses P_i harus menunggu permintaan dipenuhi.



Gambar 15 Algoritma resource allocation graph

Untuk menggambarkan algoritma ini, perhatikan *resource allocation graph* Gambar 15. Misalnya P2 meminta R2. Meskipun R2 bebas, tetapi tidak dapat dialokasikan untuk P2, karena akan menyebabkan siklus pada *graph* (Gambar 15). Siklus menandakan sistem dalam state tak aman. Jika P1 meminta R2 dan P2 meminta R1, maka terjadi *deadlock*.

I. RINGKASAN

Deadlock adalah suatu kondisi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan resource yang sedang dipakai. Karena beberapa proses itu saling menunggu, maka tidak terjadi kemajuan dalam kerja proses-proses tersebut. Deadlock dalam arti sebenarnya adalah kebuntuan. Kebuntuan yang dimaksud dalam sistem operasi adalah kebuntuan proses. Deadlock adalah masalah yang biasa terjadi ketika banyak proses yang membagi sebuah resource yang hanya boleh dirubah oleh satu proses saja dalam satu waktu. Jadi deadlock ialah suatu kondisi dimana proses tidak berjalan lagi atau pun tidak ada komunikasi lagi antar proses

J. SOAL DAN JAWABAN

1. Jelaskana bagaimana terjadinya *deadlock*
   - ⇨ *Deadlock* terjadi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan *resource* yang sedang dipakai. Karena beberapa proses itu saling menunggu, maka tidak terjadi kemajuan dalam kerja proses-proses tersebut
2. Sebutkan contoh-contoh dari *deadlock*
   - ⇨ *Proses A menggunakan CD-ROM*
   - ⇨ *Proses B menggunakan Scanner*
   - ⇨ *Proses A meminta Scanner (tanpa melepas CD-ROM) → menunggu*
   - ⇨ *Proses B meminta CD-ROM (tanpa melepas Scanner) → menunggu*
3. Sebut dan jelaskan strategi untuk menghadapi *deadlock*
   - ⇨ *Prevention (pencegahan), merupakan metode yang melakukan pencegahan terhadap 4 kondisi yang menyebabkan deadlock*
   - ⇨ *Avoidance (Penghindaran), metode dengan digunakannya informasi tambahan tentang bagaimana sumber daya diminta*
   - ⇨ *Detection (pendeteksian), Untuk mengetahui ada tidaknya deadlock dalam suatu graf dapat dilihat dari perputaran dan resource yang dimilikinya*

4. Jelaskan dan sebutkan langkah-langkah dari algoritma *banker*
   - ⇨ *Algoritma banker adalah algoritma untuk mencegah terjadinya deadlock dengan memutuskan apakah menyetujui atau menunda permintaan sumber daya oleh proses.*
   - ⇨ *Langkah-langkah algoritma banker:*
     1) Langkah pertama dari algoritma *banker* adalah cek *resource need* nya
     2) Jika *available* lebih besar atau sama dengan *resource need* maka *available* dapat dipinjamkan untuk di eksekusi. Lalu mengubah nilai available selanjutnya menjadi *available* yang dipinjamkan ditambah dengan *allocation proses* (Urutan ditulis)
     3) Jika *available* lebih kecil dari *resource need* maka tidak ada pengubahan nilai *available* atau dilewati (Tidak ditulis nomor urutannya dan proses dilanjutkan lagi)
     4) Jika sudah melewati seleksi itu cek apakah masih ada proses yang masih belum dilaksanakan/dilewati. Jika masih ada maka akan diproses lagi dari langkah awal, jika tidak ada maka akan selesai.

BAB 7
MANAJEMEN MEMORI

Manajemen Memori merupakan salah satu komponen yang ada pada sistem operasi modern saat ini. Dengan Manajemen Memori, sistem operasi dapat mengelola sumber daya secara efektif dan efisien sesuai kebutuhan program yang dijalankan. Pengelolaan memori disini tidak hanya perangkat keras penyimpan saja, melainkan juga proses, metode, dan perangkat lunak yang berhubungan dengan pengelolaan memori oleh sistem operasi. Secara perangkat keras, memori merupakan alat penyimpan data baik secara permanen maupun sementara dimana berfungsi untuk membantu jalannya suatu program komputer. Dengan tersedianya sumber daya memori yang terbatas pada sistem komputer, sedangkan program-program yang berjalan pada komputer sangat kompleks, dengan pengelolaan sistem operasi memori tersebut dapat dimaksimalkan pemanfaatannya. Terdapat berbagai teknik dalam mengelola memori secara logis berdasarkan konsep *address binding*, *overlays*, dan *swapping*. Teknik alokasi ruang secara fisik pada memori untuk menyimpan data pun juga memiliki beragam metode untuk memperoleh hasil yang efektif dan efisien. Metode alokasi memori terbagi menjadi dua meliputi alokasi memori bersebalahan dan alokasi memori tidak bersebelahan. Keduanya memiliki kelebihan dan kelamahan masing-masing.

Capaian:

1. Mampu menjelaskan pengelolaan memori oleh sistem operasi baik yang berkaitan dengan perangkat keras maupun perangkat lunak.
2. Mampu mendeskripsikan bagaimana memori dialokasikan baik menggunakan metode secara berurutan maupun tidak berurutan.

A. Memori Fisik dan Memori Logis

Sebelum menginjak ke penjelasan inti, perlu adanya pemahaman konteks dalam memahami bagaimana memori berjalan pada sistem komputer. Pada literatur ataupun sumber lainnya yang membahas tentang memori, sering dijumpai kata memori fisik atau memori logis. Dua kata tersebut memiliki

tujuan yang sama yaitu merujuk pada pembahasan memori, namun memiliki sudut pandang yang berbeda yaitu secara fisik dan secara logis.

Memori fisik merujuk pada perangkat memori secara fisik yang terpasang pada sistem komputer. Memori fisik disini bisa merujuk pada memori utama seperti *random access memory* (RAM), *cache* atau register dimana secara fisik dan nyata memiliki spesifikasi teknis tertentu. Misalkan memori RAM dengan spesifikasi kapasitas 4GB, maka secara nyata fisik RAM memiliki kapasistas untuk menyimpan data sebesar 4GB. Adapun bagaimana cara pengalokasian, pembagian, atau pengelolaan ruang memori pada RAM tersebut sudah menjadi tanggung jawab sistem operasi komputer dimana RAM berada.

Sedangkan cara pengalokasian RAM oleh komputer inilah yang menjadi konteks pada memori logis. Memori logis mengacu pemetaan penyimpanan memori fisik oleh sistem komputer. Karena pemetaan dikelola oleh sistem operasi pada komputer, aktivitas pemetaan dilakukan secara sinyal digital sehingga disebut logis (*logic*). Memori logis merupakan memori virtual yang di-*generate* oleh CPU selama masa eksekusi proses pada sebuah program.

B. Address Binding

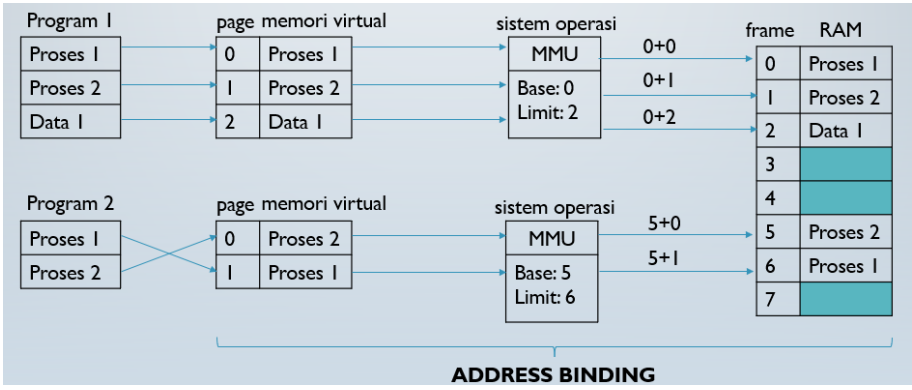
Memori logis dan memori fisik sangat berkaitan erat dalam penyimpanan. Dalam penyimpanan data tentunya terdapat label identitas agar bisa diakses yaitu alamat memori. Kaitannya dengan memori fisik dan memori logis, pengalamatan memori juga terpecah menjadi pengalamatan fisik dan pengalamatan logis. Pengalamatan fisik mengacu identitas ruang penyimpanan pada memori fisik, sedangkan pengalamatan logis mengacu pada pemetaan pengalamtan memori fisik oleh komputer. Pemetaan pada alamat memori fisik dan logis ini diterapkan pada teknologi *address binding*. *address binding* adalah proses pemetaan alamat memori logis pada alamat memori fisik dan sebaliknya. MMU (*Memory anagement Unit*) merupakan penerapan a*ddress binding* pada

sistem operasi yang bertugas untuk mengelola pemetaan alamat memori logis ke memori fisik ini

Terdapat 3 cara dalam melakukan *address binding*:

- *Compile Time Binding*: pemetaan ruang penyimpanan dari fisik menuju logis atau sebaliknya pada saat mengkompilasi program.
- *Load Time Binding*: pemetaan ruang penyimpanan dari fisik menuju logis atau sebaliknya pada saat memuat program.
- *Execution Time Binding*: pemetaan ruang penyimpanan dari fisik menuju logis atau sebaliknya pada saat eksekusi program.

Dengan adanya *address binding* oleh MMU akan memudahkan komputer dalam memanfaatkan memori semaksimal mungkin. Ruang penyimpanan pada memori bisa digunakan untuk lebih dari satu program dimana kuncinya ada pada MMU sebagai pengelola pemetaan alamat dari fisik ke logis atau sebaliknya. MMU memanfaatkan fungsi base dan limit untuk menentukan pemetaan alamat secara tepat dari alamat logis menuju alamat fisik memori dan sebaliknya. Fungsi base digunakan untuk menentukan label alamat awal sedangkan fungsi limit untuk menentukan batas alamat yang digunakan sesuai kebutuhan memori program.

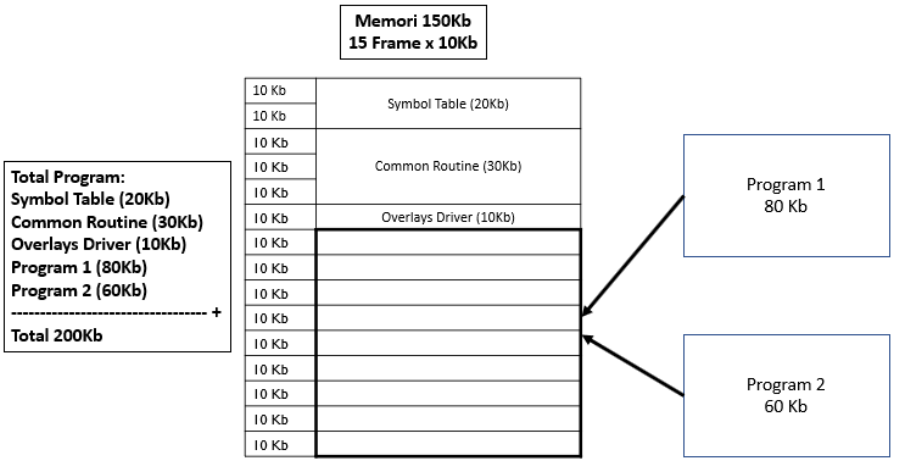


Gambar 16 Ilustrasi address binding pada MMU sistem operasi

C. *Overlays*

Overlays merupakan cara untuk memanfaatkan ruang memori yang meskipun sedikit masih tetap bisa digunakan oleh program yang membutuhkan ruang memori cukup besar. Inti dari teknologi *overlays* adalah menggunakan ruang memori secara bergantian. Diperlukan *driver overlays* untuk mengatur proses pergantian penggunaan ruang memori sehingga tidak memerlukan bantuan sistem operasi. Sistem operasi hanya cukup memberikan sinyal izin rutin berupa *syscall overlays driver* untuk bertindak secara otomatis.

Driver overlays disimpan menjadi satu dengan *syscall* sistem operasi. Dengan adanya *driver overlays*, teknik *overlays* bisa diterapkan oleh sistem operasi sehingga bisa memanfaatkan ruang memori yang terbatas untuk digunakan program secara bergantian. Seperti pada ilustrasi, memori dengan kapasistas 150Kb dapat menjalankan program dengan kapasitas 200Kb karena penggunaan ruang sebesar 90Kb secara bergantian untuk dua program.



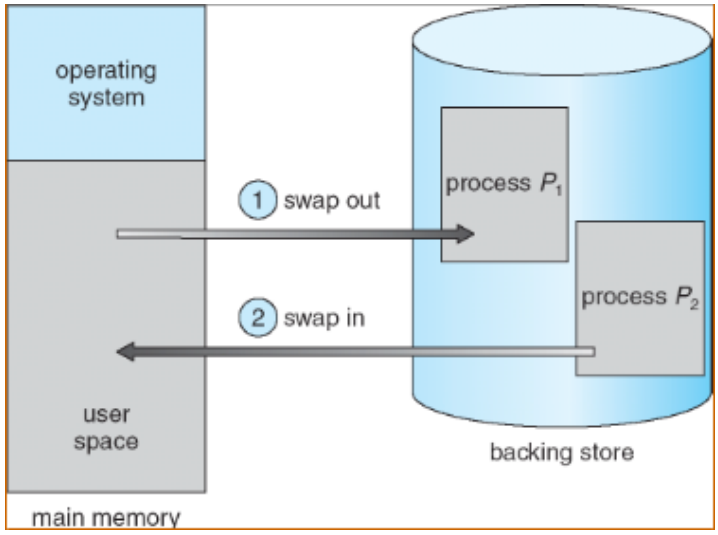
Gambar 17 Ilustrasi pemanfaatan overlays

D. *Swapping*

Apabila *overlays* bertugas untuk pengalokasian ruang memori yang dimanfaatkan bersama oleh lebih dari dua program dengan cara penggunaan bergantian, sedangkan *Swapping* merupakan teknologi dalam proses pergantian data pada

memori. *Swapping* merupakan tugas sistem operasi untuk memindah proses atau data yang datang atau keluar dari memori sekunder menuju memori utama maupun sebaliknya. Memori sekunder atau yang disebut *secondary storage* merupakan perangkat penyimpan secara permanen seperti *harddisk*, *flashdisk*, CD/DVD, atau *solid state disk*.

Apabila suatu proses atau data keluar dari memori utama menuju ke memori sekunder maka disebut dengan *swap out*, sedangkan apabila suatu proses atau data masuk ke memori utama dari memori sekunder maka disebut dengan *swap in*. proses pergantian penggunaan memori utama ini tentunya berdasarkan kebutuhan pengelolaan memori oleh sistem operasi.



Gambar 18 Ilustrasi proses *swapping*

E. Alokasi Memori Berurutan

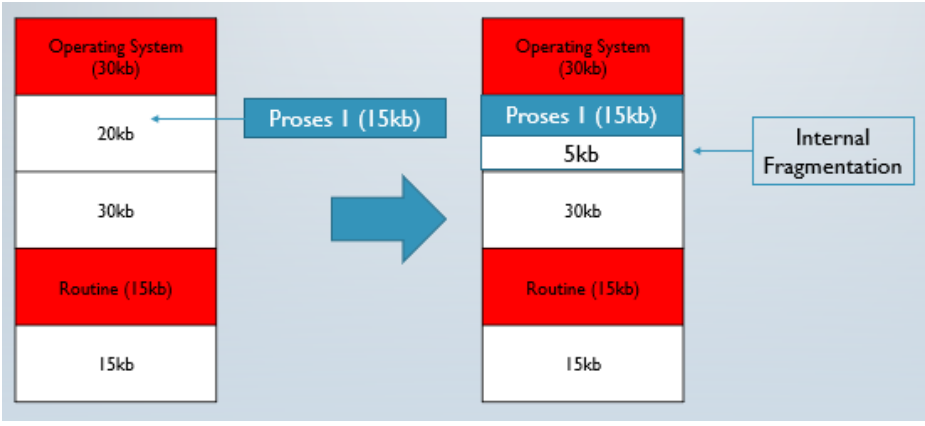
Alokasi Memori Berurutan atau *contiguous memory alocation* merupakan teknik pengalokasian ruang memori secara beruntun sesuai urutan program / proses / data yang mengantri dan dilakukan secara bergantian. Teknik pengalokasian memori secara berurutan ini merupakan metode paling lama digunakan dalam mengalokasikan ruang memori. Alokasi Memori Berurutan merupakan metode klasik yang dapat diterapkan secara mudah oleh sistem operasi

dalam mengelola memori. Terdapat dua jenis alokasi memori berurutan meliputi *Multiprogramming with Fixed Tasks* (MFT) dan *Multiprogramming with Variable Tasks* (MVT).

- MFT melakukan alokasi ruang kosong pada memori dengan ukuran kapasitas berbeda di setiap partisi namun tidak berubah (*fixed*).
- MVT memungkinkan blok proses / data bisa berpindah dari satu partisi ke partisi lain sehingga ukura kapasitas ruang kosong bisa berubah (lebih dinamis dibandingkan MFT).

F. MFT

Pada pengalokasian memori MFT, kapasistas ruang kosong pada memori tidak berubah (*fixed*). Dalam satu partisi ruang kosong hanya bisa diisi oleh satu blok proses atau data. Kelebihan dari metode MFT adalah prosesnya yang sederhana sehingga mudah untuk diterapkan. Adapun kekurangan yang terjadi adalah dimungkinkan adanya *internal fragmentation* dimana blok proses/data yang ukurannya lebih kecil dari ruang kosong yang dialokasikan akan menyebabkan terjadinya ruang kosong yang tidak bisa dimanfaatkan oleh blok proses/data lain. Hal tersebut membuat efisiensi kinerja MFT menjadi berkurang.

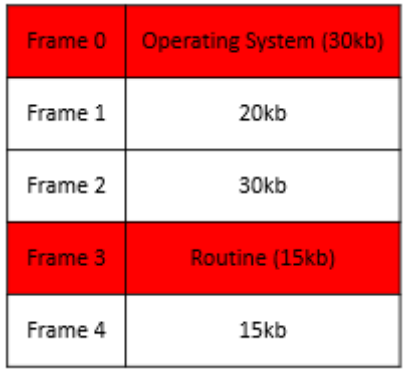


Gambar 19 Ilustrasi terjadinya *internal fragmentation* pada MFT

Alokasi memori berkesinambungan tipe MFT menggunakan algoritma agar penggunaan memori sesuai dengan urutan. Adapun algoritma MFT adalah sebagai berikut.

- *First Fit* – mengalokasikan blok proses/data pada ruang kosong berukuran cukup yang pertama kali ditemukan.
- *Best Fit* – mengalokasikan blok proses/data pada ruang kosong yang ukuran paling sesuai.
- *Next Fit* – mengalokasikan blok proses/data pada ruang kosong berukuran cukup yang pertama kali ditemukan namun dimulai dari akhir alamat frame.
- *Worst Fit* – mengalokasikan blok proses/data pada ruang kosong yang memiliki ukuran paling besar.

Untuk memahami bagaimana algoritma dari metode MFT berjalan maka digunakan studi kasus kondisi memori seperti berikut. Diketahui terdapat memori utama berkapasitas 110Kb dengan pembagian 5 ruang frame meliptui frame 0 berkapasistas 30Kb, frame 1 20Kb, frame 2 30Kb, frame 3 15Kb, dan frame 4 15 Kb. Kebetulan frame 0 dan frame 3 telah penuh terisi blok data sistem operasi dan routine.



Gambar 20 Ilustrasi kondisi memori berkapasitas 110kb

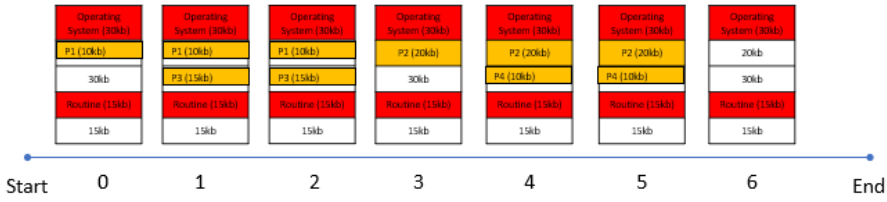
Kemudian secara beruntun terdapat 5 proses yang terdiri Proses 1, Proses 2, Proses 3, Proses 4, dan Proses 5 dimana kelima proses tersebut datang dengan kondisi kedatangan, ukuran, dan waktu eksekusi yang berbeda-beda seperti pada tabel berikut.

Tabel Kondisi Antrian Proses

Proses	Ukuran (Kb)	Waktu Kedatangan	Waktu Eksekusi
P1	10	0	2
P2	20	2	2
P3	15	1	1
P4	10	4	1

1 Algoritma First Fit

Algoritma *first fi*t yaitu mengalokasikan blok proses/data pada ruang kosong berukuran cukup yang pertama kali ditemukan. Sehingga berdasarkan studi kasus yang dijelaskan sebelumnya maka diperoleh langkah-langkah alokasi memori seperti berikut.



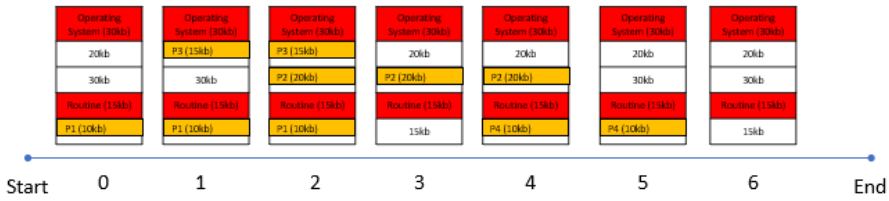
Gambar 21 Ilustrasi alokasi memori MFT dengan algoritma *first fit*

- Waktu ke-0: P1 dengan waktu kedatangan di waktu ke-0 berada pada frame 1 karena ruang telah cukup saat pertama kali.
- Waktu ke-1: P1 masih berada pada frame 1 karena waktu eksekusi belum habis. P3 datang dengan kedatangan di waktu ke-1 menempati frame 2 karena ruang telah cukup saat pertama kali.
- Waktu ke-2: P2 datang di waktu ke-2, namun ruang kosong memori tidak ada yang cukup sehingga menunggu. Kondisi masih sama seperti waktu ke-1.

- Waktu ke-3: P1 dan P3 selesai dieksekusi sehingga ruang banyak kosong. P2 yang telah menunggu, menempati frame 1 karena ruang telah cukup saat pertama kali.
- Waktu ke-4: P2 masih berada pada frame 1 karena waktu eksekusi belum habis. P4 datang dengan kedatangan di waktu ke-4 menempati frame 2 karena ruang telah cukup saat pertama kali.
- Waktu ke-5: kondisi masih sama seperti waktu ke-4.
- Waktu ke-6: P2 dan P4 selesai dieksekusi sehingga ruang kosong.

2 Algoritma Best Fit

Algoritma *Best Fit* yaitu mengalokasikan blok proses/data pada ruang kosong yang ukuran paling sesuai. Sehingga berdasarkan studi kasus yang dijelaskan sebelumnya maka diperoleh langkah-langkah alokasi memori seperti berikut.



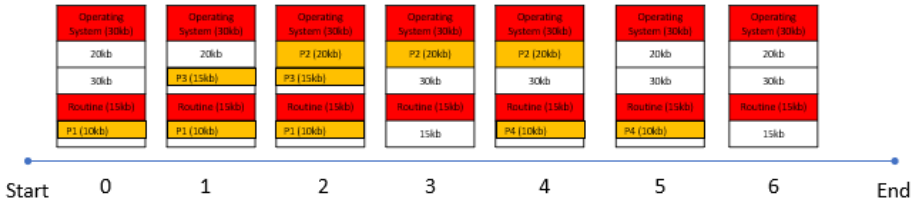
Gambar 22 Ilustrasi alokasi memori MFT dengan algoritma *best fit*

- Waktu ke-0: P1 datang di waktu ke-0 dan menempati frame 4 karena ukurannya paling sesuai.
- Waktu ke-1: P1 masih berada pada frame 4 karena waktu eksekusi belum habis. P3 datang di waktu ke-1 dan menempati frame 1 karena ukurannya paling sesuai.
- Waktu ke-2: P1 dan P3 masih berada pada frame sebelumnya karena waktu eksekusi belum habis. P2 datang di waktu ke-2 dan menempatti frame 2 karena ukurannya paling sesuai.
- Waktu ke-3: P1 dan P3 sudah selesai dieksekusi dan tinggal P2 masih di frame 2 karena waktu eksekusi belum habis.

- Waktu ke-4: P2 masih di frame 2 karena waktu eksekusi belum habis. P4 datang di waktu ke-4 dan menempati frame 4 karena ukurannya paling sesuai.
- Waktu ke-5: P4 masih di frame 4 karena waktu eksekusi belum habis.
- Waktu ke-6: P4 selesai dieksekusi sehingga ruang kosong.

3 *Algoritma Next Fit*

Algoritma Next Fit yaitu mengalokasikan blok proses/data pada ruang kosong berukuran cukup yang pertama kali ditemukan namun dimulai dari akhir alamat frame. Sehingga berdasarkan studi kasus yang dijelaskan sebelumnya maka diperoleh langkah-langkah alokasi memori seperti berikut.



Gambar 23 Ilustrasi alokasi memori MFT dengan algoritma *next fit*

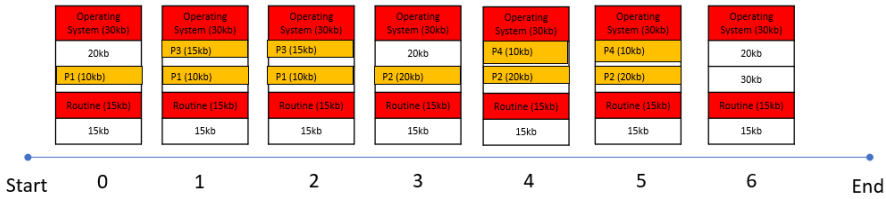
- Waktu ke-0: P1 datang di waktu ke-0 dan menempati frame 4 karena ruang telah cukup saat pertama kali datang dari frame terbawah.
- Waktu ke-1: P1 masih berada pada frame 4 karena waktu eksekusi belum habis. P3 datang di waktu ke-1 dan menempati frame 2 karena ruang telah cukup saat pertama kali dari frame terbawah.
- Waktu ke-2: P1 dan P3 masih berada pada frame sebelumnya karena waktu eksekusi belum habis. P2 datang di waktu ke-2 dan menempati frame 1 karena ruang telah cukup saat pertama kali dari frame terbawah.
- Waktu ke-3: P1 dan P3 selesai dieksekusi sehingga ruang banyak kosong. P2 masih menempati frame 1 karena waktu eksekusi belum habis.
- Waktu ke-4: P2 masih berada pada frame 1 karena waktu eksekusi belum habis. P4 datang dengan kedatangan di waktu

ke-4 menempati frame 4 karena ruang telah cukup saat pertama kali dari frame terbawah.

- Waktu ke-5: P2 selesai dieksekusi dan P4 masih berada pada frame yang sama.
- Waktu ke-6: P4 selesai dieksekusi sehingga ruang kosong.

4 *Algoritma Worst Fit*

Algoritma Worst Fit yaitu mengalokasikan blok proses/data pada ruang kosong yang memiliki ukuran paling besar. Sehingga berdasarkan studi kasus yang dijelaskan sebelumnya maka diperoleh langkah-langkah alokasi memori seperti berikut.



Gambar 24 Ilustrasi alokasi memori MFT dengan algoritma *worst fit*

Ilustrasi Alokasi Memori MFT dengan Algoritma Worst Fit

- Waktu ke-0: P1 datang di waktu ke-0 dan menempati frame 2 karena kapasitas frame paling besar dibandingkan frame kosong yang lain.
- Waktu ke-1: P1 masih berada di frame yang sama karena waktu eksekusi belum selesai. P3 datang di waktu ke-1 dan menempati frame 1 karena kapasitas frame paling besar dibandingkan frame kosong yang lain.
- Waktu ke-2: P2 datang di waktu ke-2 namun ruang kosong memori tidak ada yang cukup sehingga menunggu. Kondisi memori seperti waktu ke-1.
- Waktu ke-3: P1 dan P3 selesai dieksekusi sehingga banyak ruang kosong. P2 menempati frame 2 karena kapasitas frame paling besar dibandingkan frame kosong yang lain.
- Waktu ke-4: P2 masih berada di frame yang sama karena waktu eksekusi belum selesai. P4 datang di waktu ke-4 dan

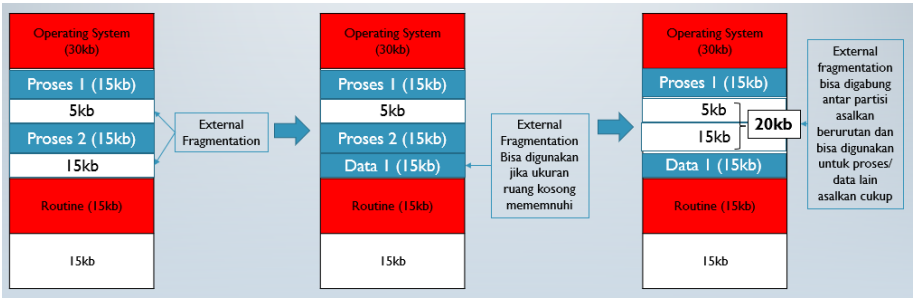
menempati frame 1 karena kapasitas frame paling besar dibandingkan frame kosong yang lain.

- Waktu ke-5: P2 dan P4 masih berada di frame yang sama seperti di waktu ke-4 karena waktu eksekusi belum selesai
- Waktu ke-6: P2 dan P4 selesai dieksekusi dan ruang memori kosong.

G. MVT

MVT memungkinkan proses / data bisa berpindah dari satu partisi ke partisi lain dimana ukuran ruang kosong memori bisa menyesuaikan. Hal ini membuat teknik *address binding* dilakukan secara dinamis. Kelebihan dari MVT adalah penggunaan ruang kosong pada memori secara dinamis. Kekurangannya adalah penerapan yang kompleks sehingga mahal apabila diterapkan. MVT memungkinkan terjadi *external fragmentation. External fragmentation* adalah terjadinya ruang kosong karena tidak terisi oleh blok proses / data antar ruang kosong. Namun terjadinya *external fragmentation* membuat antar fragmen eksternal bisa digabungkan dan digunakan untuk memenuhi kebutuhan proses / data lain asalkan cukup dan berurutan.

MVT memungkinkan dua blok proses / data masuk dalam sebuah ruang kosong memori secara bersamaan apabila memang cukup. Pabila blok proses / data yang berada pada awal alamat tersebut selesai dieksekusi terlebih dahulu, mengakibatkan *external fragmentation* pada partisi tersebut dengan partisi ruang kosong lainnya. Antara fragmen eksternal partisi ruang kosong yang satu dengan fragmen eksternal ruang kosong yang lainnya bisa digabung pada metode MVT asalkan berurutan. Penggabungan *external fragmentation* ini mengakibatkan partisi ruang kosong baru pada memori dimana bisa digunakan untuk blok program / proses lainnya.



Gambar 25 Ilustrasi terjadinya *external fragmentation* pada MVT

H. Alokasi Memori Tidak Berurutan

Alokasi Memori Tidak Berurutan atau *Non-contiguous allocation memory* memungkinkan untuk mengelola pengalokasian ruang memori secara tidak berurutan. Hal ini bertujuan agar memaksimalkan ruang kosong seberapapun ukurannya. Dengan adanya pengalokasian tidak berurut, penyimpanan blok data / proses bisa dinamis *mengikuti* dimana ruang kosong yang ada. Namun demikian, dengan adanya kelebihan bahwa efisiensi penggunaan memori tinggi, namun mengakibatkan lamanya waktu ekskusi karena harus ada pemetaan alamat. Kunci berhasilnya blok proses / data bisa dikumpulkan kembali dari ruang kosong memori yang terpisah-pisah adalah adanya pemetaan alamat.

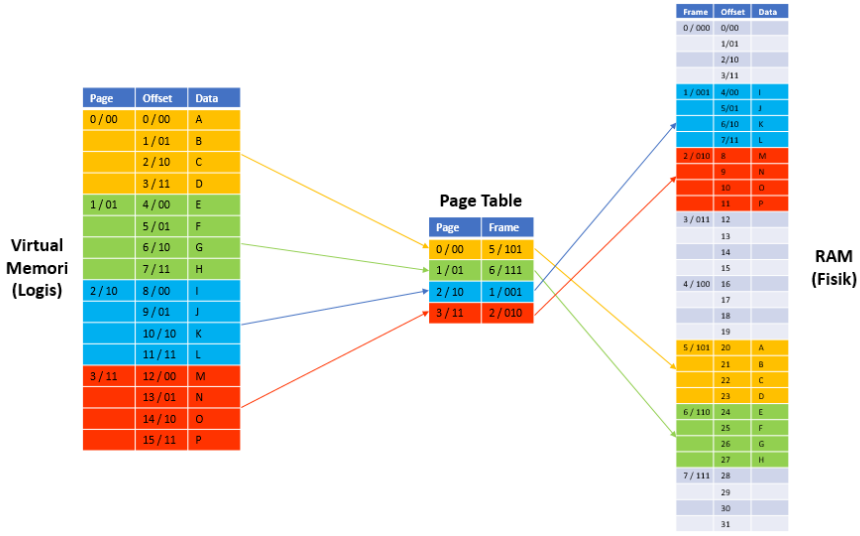
Sehingga pada alokasi memori Tidak Berurutan ini memang benar-benar mememrlukan peran *address binding* seperti MMU untuk memenuhi pemetaan antara memori fisik dan memori logis. Bisa saja pada suatu program, memori logis sebagai memori virtual dikelola secara berurutan namun pada kenyataannya, data disimpan secara tidak berurutan pada memori fisik. Semua tergantung pada kinerja MMU. Terdapat dua pendekatan pemetaan pengalamatan meliputi teknik *Paging* dan *Segmentasi*.

I. Paging

Paging dilakukan dengan membagi memori ke dalam beberapa blok, jika pada memori logis blok disebut dengan *page*, sedangkan pada memori fisik blok disebut dengan *frame*. Standarnya memori fisik dibagi ke dalam blok *frame* yang berukuran antara 512b sampai dengan 16Kb setiap

frame-nya. Dengan membagi memori fisik menjadi blok-blok *frame* bertujuan memudahkan pengguna dalam menyimpan data dan mengakses proses. Bedasarkan sudut pandang pengguna, penyimpanan terlihat berurutan dimana dipresentasikan oleh memori logis, padahal kenyataannya penyimpanan pada memori fisik dilakukan secara menyebar sesuai kondisi. Kemudahan tersebut diterapkan dengan membuat *page table* sebagai pemetaan data dari memori logis ke memori fisik dan sebaliknya (tentunya dibantu dengan MMU).

Penerapan teknik *Paging* memang kompleks namun menghasilkan efisiensi yang tinggi dalam memanfaatkan ruang kosong memori. Dalam penerapannya diperlukan teknik untuk menghitung kapasitas *page* dan *frame* menggunakan angka *biner* sebagai pengalamatannya. Hal ini dikarenakan sistem komputer sendiri yang hanya dapat memahami angka *biner*. Dimisalkan alamat memori logis (*logical address*) sebesar 4 bit, dibagi menjadi dua bagian meliputi 2bit pertama untuk alamat (label) *page* dan 2bit terakhir untuk alamat *offset*. Karena panjang alamat logis sebesar 4bit maka akan diperoleh 16 *page* dengan ukuran kapasitas per page tergantung pengelolaan oleh sistem operasi. Sedangkan untuk panjang alamat memori fisik (*physical address*) sebesar 5 bit, dibagi menjadi dua bagian juga yaitu 3bit pertama untuk alamat frame dan 2bit terakhir untuk alamat offset. Karena panjang alamat fisik 5bit maka terdapat 32 frame dengan besar kapasitas tergantung sistem operasi. Panjang alamat offset antara alamat memori logis dengan alamat memori fisik harus sama agar pemetaan alamat bisa dilakukan. Dengan bantuan tabel page maka antara memori logis dengan memori fisik bisa dipetakan seperti contoh ilustrasi berikut.



Gambar 26 Ilustrasi sederhana alokasi memori dengan *paging*

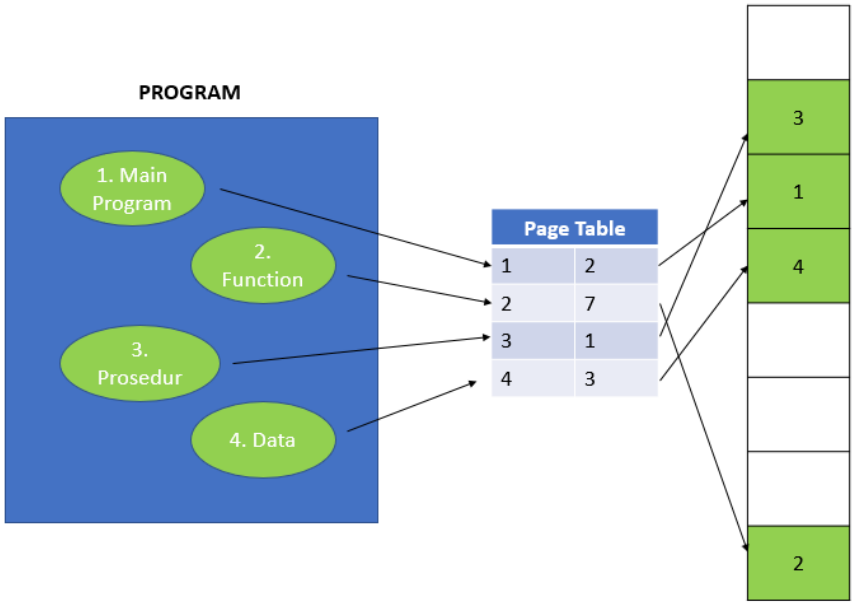
Pada memori logis terdapat empat *page* dimana per *page* terdapat 4 ruang *offset*. Setiap ruang offset akan disimpan sebuah data. Pada tabel page, alamat page akan dipetakan dengan alamat frame, dan kondisi pemetaan akan bersifat *random* sesuai frame mana yang kosong. Pada memori fisik terdapat 8 frame dimana setiap frame-nya terdapat 4 ruang offset. Setiap ruang offset akan disimpan sebuah data juga seperti kaidah pada memori logis. Hal ini lah yang membuat panjang alamat offset baik pada memori logis maupun memori fisik harus sama.

J. Segmentasi

Dengan adanya teknik *paging* saja tidak cukup untuk *memudahkan* pengguna. Karena saat ini, ketika pengguna melakukan eksekusilebih dari dua program pada komputer, tentunya pada level proses / data ada sebuah proses/data yang dibutuhkan oleh dua program. Hal ini akan boros ruang penyimpanan apabila setiap program harus menyalin proses/data yang sama dan disimpan pada ruang penyimpanan yang berbeda. Untuk menghemat memori semaksimal mungkin, maka terdapat ide untuk berbagi proses / data yang dibutuhkan oleh dua atau lebih program yang berbeda tanpa harus disalin atau diduplikat. Dengan demikian

penyimpanan proses/data tersebut bisa lebih hemat karena tidak banyak salinannya.

Dengan teknik segmentasi maka dimungkinkan untuk sistem operasi mengelola alokasi ruang pada memori berdasarkan melakukan segmentasi terhadap program. Program yang sebelumnya dianggap sebagai satu blok proses/data dan disimpan di memori dalam satu ruang yang tetap (*fixed*), kali ini diubah menjadi beberapa segmen agar dapat disimpan secara terpisah dalam memori. Sehingga akan dimungkinkan satu blok segmen proses bisa digunakan untuk proses eksekusi program lain. Segmentasi pada program bisa dilakukan dengan berbagai cara tergantung sistem operasi mengelola program, contoh dibagi menjadi program utama (*main program*), fungsi, prosedur, data, dan lainnya.



Gambar 27 Ilustrasi segmentasi pada program

Dalam alokasi memori Segmentasi, penggunaan paging tetap ada dan tetap memerlukan tabel page. Namun setiap program yang dieksekusi akan mengalami segmentasi menjadi bagian-bagian modular. Tipe blok yang murni proses atau hanya data saja tentunya akan bisa digunakan oleh lebih dari dua program dengan adanya segmentasi.

K. RINGKASAN

Manajemen memori bertujuan untuk memaksimalkan pemanfaatan sumber daya memori oleh sistem operasi. Dalam mengelola memori, sistem operasi memiliki teknik *Address* Binding untuk pemetaan memori fisik ke memori logis dan sebaliknya, teknik Overlays untuk penggunaan ruang memori secara bergantian, dan teknik Swapping untuk memindah data / proses keluar masuk memori, terutama dari memori skunder ke memori utama (dan sebalikny). Dalam mengalokasikan ruang memori, sistem operasi bisa menggunakan teknik Alokasi Memori Berurutan atau Alokasi Memori Tidak Berurutan. Teknik Alokasi Memori Berurutan merupakan metode alokasi memori paling lama digunakan karena sederhana dan mudah diterapkan

Teknik Alokasi Memori Berurutan dibagi lagi menjadi dua metode meliputi MFT dan MVT. MFT untuk alokasi ruang memori yang dikondisikan tetap (*fixed*) sedangkan MVT untuk alokasi ruang memori secara dinamis. Cara dalam menempatkan blok proses / data pada ruang memori dengan alokasi MFT dibagi menjadi empat algoritma meliputi First Fit, Best Fit, Next Fit, dan Worst Fit. Untuk teknik Alokasi Memori Tidak Berurutan maka cara alokasi secara dasar menggunakan metode Paging dan Segmentasi. Metode Paging memanfaatkan ruang kosong secara *random* sehingga diperlukan pemetaan menggunakan tabel paging antara alamat memori logis dengan alamat memori fisik. Sedangkan metode Segmentasi, merupakan metode untuk membagi program menjadi segmen sehingga dapat disimpan ke dalam memori sesuai dengan ruang kosong yang ada.

L. LATIHAN

Sekumpulan proses diketahui memiliki kondisi seperti pada tabel, dan akan disimpan dalam memori utama dengan kondisi awal sebagai berikut.

Proses	Size (Kb)	Burst Time
P0	100	3
P1	10	1
P2	35	2
P3	15	1
P4	23	2
P5	6	1
P6	25	1

Memori
50 Kb
200 Kb
70 Kb
115 Kb
15 Kb

Tunjukkan langkah bagaimana proses dialokasikan menggunakan teknik Alokasi Memori Berurutan MFT dengan algoritma:

1) First Fit
2) Best Fit
3) Next Fit
4) Worst Fit

BAB 8
SISTEM FILE

File komputer merupakan sumber daya komputer yang digunakan untuk merekam data secara diskrit dalam perangkat penyimpanan komputer. Seperti kalimat yang dapat ditulis pada kertas, demikian juga informasi dapat ditulis pada file komputer. Terdapat banyak jenis file komputer yang dirancang untuk tujuan berbeda. File dapat dirancang untuk menyimpan video, gambar, program komputer, pesan tertulis, atau berbagai jenis data lainnya serta beberapa jenis file dapat menyimpan jenis informasi yang berbeda sekaligus. Dengan memanfaatkan program komputer, pengguna dapat membaca, menutup, membuka, dan mengubah file komputer. File komputer dapat dimodifikasi, dibuka kembali, dan disalin beberapa kali. Pengertian file input/output menurut Lanny Johan "perekaman data ke dalam media rekam, pembacaan data dari media rekam itu dan penghapusan data."

A. Menguasai Input Output File

Dalam pemrograman komputer, aliran standar adalah saluran komunikasi input dan output yang telah terhubung sebelumnya antara program komputer dan lingkungannya saat program tersebut mulai dieksekusi. Tiga koneksi input/output (I/O) disebut input standar (*stdin–keyboard*), output standar (*stdout* –awalnya printer) dan kesalahan standar (*stderr*–monitor). *Streaming* dapat dialihkan ke perangkat dan/atau file lain. Di lingkungan saat ini, *stdout* biasanya diarahkan ke monitor.

File komputer disimpan pada perangkat penyimpanan sekunder dan digunakan untuk memelihara data program dari waktu ke waktu. Sebagian besar bahasa pemrograman memiliki fungsi atau pustaka bawaan untuk mendukung pemrosesan file sebagai aliran teks. Perlu memahami cara membuka, membaca, menulis, dan menutup file teks. Berikut penjelasan istilah File Input/Output:

File Teks – File yang terdiri dari karakter dari set kode karakter ASCII. File teks (juga dikenal sebagai file teks ASCII) berisi data karakter. Saat membuat file teks, biasanya

menganggapnya terdiri dari serangkaian baris. Pada setiap baris terdapat beberapa karakter (termasuk spasi, tanda baca, dll.) dan kami biasanya mengakhiri baris dengan pengembalian (karakter dalam rangkaian kode karakter ASCII). Pengembalian juga dikenal sebagai karakter baris baru. Anda kemungkinan besar sudah terbiasa dengan kode pelarian \n yang digunakan dalam banyak bahasa pemrograman untuk menunjukkan karakter kembali saat digunakan dalam string literal.

1. *Filename* – Nama dan ekstensinya. Sebagian besar sistem operasi memiliki batasan karakter mana yang dapat digunakan dalam nama file. Contoh Lab_05.txt. Karena beberapa sistem operasi tidak mengizinkan spasi, gunakan garis bawah jika diperlukan untuk spasi dalam nama file.
2. *Path* (*Filespec*) – Lokasi file beserta nama filenya. *Filespec* adalah kependekan dari spesifikasi file. Sebagian besar sistem operasi memiliki seperangkat aturan tentang cara menentukan drive dan direktori (atau jalur melalui beberapa tingkat direktori) bersama dengan nama file. Contoh: C:\myfiles\cosc_1436\Lab_05.txt. Karena beberapa sistem operasi tidak mengizinkan spasi, gunakan garis bawah jika diperlukan saat membuat folder atau sub-direktori.
3. Buka – Program meminta sistem operasi untuk mengizinkannya mengakses file yang ada atau membuka file baru. Dalam sebagian besar bahasa pemrograman saat ini, tipe data file ada dan digunakan untuk pemrosesan file. Variabel file akan digunakan untuk menyimpan token perangkat yang diberikan sistem operasi ke file yang sedang dibuka. Fungsi atau metode terbuka digunakan untuk mengambil token perangkat, dan biasanya memerlukan setidaknya dua parameter: jalur dan mode (baca, tulis, tambahkan atau kombinasinya). *Pseudocode* yang sesuai adalah:
   - Baca – Memindahkan data dari perangkat yang telah dibuka ke lokasi memori yang ditentukan dalam program.
   - Tutup – Program meminta sistem operasi untuk merilis file yang sebelumnya dibuka. Ada dua alasan untuk menutup file. Pertama, melepaskan file dan membebaskan sumber daya sistem operasi terkait. Kedua, jika menutup file yang

dibuka untuk output; itu akan menghapus *buffer* sistem operasi dan memastikan bahwa semua data disimpan secara fisik dalam file output.

B. Algoritma Alokasi File

Ruang untuk menyimpan berkas pada tempat penyimpanan utama, dalam hal ini memory, tidak cukup besar untuk menampung berkas dalam jumlah besar karena ukurannya yang terbatas dan harganya yang mahal, memory hanya dapat menyimpan berkas saat komputer dalam keadaan terhubung dengan arus listrik. Karena hal-hal tersebut di atas, maka diperlukan tempat penyimpanan sekunder (disk) yang dapat mempertahankan berkas walaupun tidak ada arus listrik yang mengalir ke komputer. Kemudahan dalam mengakses langsung suatu disk memberikan fleksibilitas dalam membantu mengimplementasikan sebuah berkas.

Masalah utama dalam implementasi adalah bagaimana mengalokasikan berkas-berkas ke dalam disk, sehingga disk dapat terimplementasi dengan efektif dan berkas dapat diakses dengan cepat. Dalam bab ini, akan dijelaskan bagaimana caranya mengalokasikan berkas untuk sebuah berkas agar disk dapat digunakan secara optimal dan agar berkas dapat diakses secara cepat. Juga akan dibahas, beserta cara mengatur ruang kosong pada disk dan cara mempertahankan kekonsistenan berkas serta mencegah kehilangan data. Metode alokasi berhubungan dengan bagaimana blok-blok pada disk dialokasikan untuk file. Terdapat beberapa metode alokasi antara lain alokasi berurutan (*contiguous allocation*), alokasi berhubungan (*linked allocation*) dan alokasi berindeks (*indexed allocation*).

Ada 3 metode untuk mengalokasi ruang disk yang digunakan secara luas yaitu:

1. Alokasi Berkesinambungan (*Contiguous Allocation*)

   Pada alokasi berurutan, setiap file menempati sekumpulan blok yang berurutan pada disk model ini sangat sederhana karena hanya membutuhkan lokasi awal (block #) dan panjang (jumlah blok). Akses pada blok disk dilakukan secara *random* dan memakan banyak ruang (permasalahan *dynamic storage-*

allocation). File yang disimpan secara berurutan tidak dapat berkembang.

Jadi sebuah berkas didefinisikan oleh alamat disk blok pertama dan panjangnya dengan satuan blok atau beberapa blok yang diperlukannya. Bila suatu berkas memerlukan n buah blok dan blok awalnya adalah a, berarti berkas tersebut disimpan dalam blok di alamat a, a + 1, a + 2, a + 3, … , a + n - 1. Direktori mengidentifikasi setiap berkas hanya dengan alamat blok pertama berkas tersebut disimpan yang dalam contoh di atas adalah a, dan banyaknya blok yang diperlukan untuk mengalokasikan berkas tersebut yang dalam contoh di atas adalah n. Berkas yang dialokasikan dengan metode ini akan mudah diakses, karena pengaksesan alamat a + 1 setelah alamat a tidak diperlukan perpindahan head, jika diperlukan pemindahan head, maka head tersebut akan hanya akan berpindah satu track. Hal tersebut menjadikan metode ini mendukung pengaksesan secara berurutan.

2. Alokasi berhubungan (*Linked Allocation*)
   Pada alokasi berhubungan, setiap file adalah sebuah *linked list* dari blok-blok terpisah pada disk. Pada setiap blok terdapat satu pointer yang menunjuk ke blok lain.

   Alokasi berhubungan mempunyai bentuk yang sederhana, hanya memerlukan alamat awal. Sistem manajemen ruang bebas pada alokasi berhubungan tidak memakan banyak ruang. Model ini tidak menggunakan *random access*. Blok yang diakses adalah blok ke-Q pada rantai link dari blok pada file. Perpindahan ke blok = R + 1. Contoh sistem file yang menggunakan alokasi berhubungan adalah *file-allocation table* (FAT) yang digunakan MS-DOS dan OS/2. Metode ini menyelesaikan semua masalah yang terdapat pada alokasi berkesinambungan.

   - Dengan metode ini, setiap berkas merupakan *linked list* dari blok-blok disk, dimana blok-blok disk dapat tersebar di dalam disk.
   - Setiap direktori berisi sebuah penunjuk (*pointer*) ke awal dan akhir blok sebuah berkas,

- Setiap blok mempunyai penunjuk ke blok berikutnya.
- Dengan metode ini, setiap direktori masukan mempunyai penunjuk ke awal blok disk dari berkas
- Untuk membaca suatu berkas, cukup dengan membaca blok-blok dengan mengikuti pergerakan penunjuk

Metode link ini memiliki beberapa kerugian, karena petunjuk ke blok berikutnya memerlukan ruang. Bila ukuran petunjuknya 4 *byte* dari blok yang ukurannya 512 byr, berarti 0,78% dari ruang disk hanya digunakan untuk petunjuk saja. Hal ini dapat diminimalisasikan dengan menggunakan *cluster* yang menggabungkan 4 blok dalam satu *cluster*, jadi jumlah petunjuknya akan berkurang dari yang tidak memakai *cluster*. Paling penting dalam metode ini adalah menggunakan *file-allocation table* (FAT). Tabel tersebut menyimpan setiap blok yang ada di disk dan diberi nomor sesuai dengan nomor blok. Jadi, direktori hanya menyimpan alamat dari blok pertama saja, dan untuk selanjutnya dilihat dari tabel tersebut yang menunjukkan ke blok berikutnya. Jika memakai metode ini, akan menyebabkan mudahnya untuk membuat berkas baru atau mengembangkan berkas sebelumnya. Carilah alamat terakhirnya yang memiliki ciri tertentu dan ubahlah isi dari tabel tersebut dengan alamat blok penambahan. Alamat terakhir berisi hal yang unik, sebagai contoh ada yang menuliskan -1, tapi ada juga yang menuliskannya EOF (*End Of File*). Metode link yang menggunakan FAT akan mempersingkat waktu yang diperlukan untuk mencari sebuah berkas. Karena bila tidak menggunakan FAT, berarti harus ke satu blok tertentu dahulu dan baru diketahui alamat blok selanjutnya. Dengan menggunakan FAT dapat melihat alamat blok selanjutnya disaat masih menuju blok yang dimaksud. Tetapi bagaimanapun ini belum dapat mendukung pengaksesan secara langsung.

3. Alokasi Berindeks (*Indexed Allocation*)
   Pada alokasi berindeks, terdapat satu blok yang berisi *pointer* ke blok-blok file. Alokasi berindeks berupa bentuk logika.

   Pada alokasi berindeks, memerlukan tabel indeks yang membawa pointer ke blok-blok file yang lain. Akses dilakukan

secara *random*. Merupakan akses dinamis tanpa fragmentasi eksternal, tetapi mempunyai blok indeks yang berlebih. Pemetaan dari logika ke fisik dalam file ukuran maksimum 256K word dan ukuran blok 512 word hanya memerlukan 1 blok untuk tabel indeks. Apabila pemetaan dari logika ke fisik dalam sebuah file dari ukuran tak hingga (ukuran blok adalah 512 word) maka digunakan skema menghubungkan blok link dari tabel indeks (ukuran tak terbatas). Untuk ukuran file maksimum 5123 digunakan skema two-level indeks. Pada skema two-level indeks terdapat tabel indeks luar dan dalam. Indeks dipetakan ke tabel indeks luar kemudian dipetakan ke tabel indeks dalam setelah itu mengakses blok file yang dimaksud. Sistem operasi UNIX mengimplementasikan kombinasi alokasi berurutan dan alokasi berindeks

DAFTAR PUSTAKA

Andrew S. Tanenbaum. 2008. Modern Operating System. 3th Edition: Prentice Hall.

Bob DuCharme. 2001. The Operating System Handbook or, Fake Your Way Through Minis and Mainframes. Singapore: McGraw-Hill Book Co.

John Wiley & Sons. Stalling, William, 1995, Operating Sistems. New Jersey. Prentice Hall

Naghibzadeh. 2005. Operating System: Concepts and Techniques. Amazon

Pangera, Abas. 2016. Konsep Dasar Implementasi Sistem File. Yogjakarta

Ron White. 1998. How Computers Work, Fourth Edition, Que corporation, A Division of Macmillan Computer Publishing. USA.

Silberschatz, A., dan Galvin P.2003. Operating Sistem Concept. Sixth Edition. Massachhussets: Addisson

Silberschatz, Galvin, Gagne. 2009. "8th Edition Operating System Concepts". John Wiley & Sons. United States of Americ

William A. 1993. Introduction to Operationg System. New York: HarperCollins College Publishers.

Watrianhos, Ronald. 2018. Buku Ajar Sistem Operasi. Ponorogo: Uwais Inspirasi Indonesia

GLOSARIUM

Address Binding	Cara instruksi dan data (yang berada di disk sebagai file yang dapat dieksekusi) dipetakan ke alamat memori
Assignment Edge	Satu set garis berarah dari Rj ke Pi yang menunjukkan bahwa sumber daya Rj telah dialokasikan untuk memproses Pi, dan bahwa Pi saat ini memegang sumber daya Rj
Biner	Sistem penulisan angka dengan menggunakan dua simbol yaitu 0 dan 1
Bootstrap Program	Program yang terdapat pada ROM pada komputer yang dapat menempatkan kernel
Buffer	Area memori yang menyimpan data ketika mereka sedang di pindahkan antara dua device atau antara device dan aplikasi
Buffering	Hambatan saat seseorang ingin mengakses sebuah video atau konten serupa
Cache	Proses yang digunakan oleh browser dan aplikasi untuk menyimpan informasi
Claim Edge	Indikasi yang menunjukkan bahwa proses Pi dapat meminta sumber daya Rj beberapa waktu di masa depan
Command Interpreter	Program yang membaca perintah berdasarkan teks yang diterima melalui User atau File
Concurrency	Kemampuan suatu program untuk menangani multiple order atau request
Context Switch	Switching dari CPU dari satu proses atau thread yang lain
Context Switching thread	Proses penyimpanan suatu thread (urutan terkecil dari instruksi terprogram yang dapat dikelola secara independen oleh scheduler)
Crystal Quartz	Material quartz yang sangat kecil, berupa potongan tipis atau wafer kuarsa yang dipotong dengan dua permukaan paralel yang dilapisi logam untuk membuat sambungan listrik yang diperlukan. Ukuran fisik dan ketebalan sepotong kristal kuarsa dikontrol dengan ketat karena mempengaruhi frekuensi akhir atau dasar osilasi.
Device Controller	Sirkuit digital yang berfungsi mengontrol kerja komponen mekanik ataupun elektris lainnya dari piranti I/O
Dual Mode	Model ganda
Electrically Erasable Programmable Read Only Memory (EEPROM)	Sejenis chip memori tidak-terhapus yang digunakan dalam komputer dan peralatan elektronik lain untuk menyimpan sejumlah konfigurasi data pada alat elektronik tersebut yang tetap harus terjaga meskipun sumber daya diputuskan, seperti tabel kalibrasi atau kofigurasi perangkat
Error	Kesalahan
File Allocation	Sistem berkas yang menggunakan struktur tabel alokasi berkas sebagai cara dirinya beroperasi

Table	
Firmware	Perangkat yang kecil dan berada dalam perangkat lunak, ia berguna untuk membantu perangkat keras bergerak sesuai fungsinya
Fork System Call	Sistem call yang membuat suatu proses baru pada sistem operasi UNIX
Frame	Bagian pinggir atau yang disebut bingka
Gadget	Perangkat elektronik kecil yang memiliki fungsi khusus
Gant Chart	Tools management proyek yang sangat populer digunakan
Hardisk	Hardware yang biasa digunakan untuk menyimpan data di sebuah komputer atau laptop
Hardware	Perangkat Keras
Independent	Mandiri
InfiniBand	Arsitektur komunikasi berkecepatan tinggi yang bertujuan digunakan untuk alat interkoneksi, seperti server, secondary storage, dan switch jaringan
Linked List	Koleksi linear dari data, yang disebut sebagai nodes, dimana setiap node akan menunjuk pada node lain melalui sebuah pointer
Local Area Network (LAN)	Jaringan komputer yang hanya mencakup wilayah lokal saja
Memory Management Unit (MMU)	Perangkat keras yang memetakan alamat virtual ke alamat fisik.
Multicoreprocessor	Processor komputer yang terdiri dari dua chip atau lebih memiliki unit pengolah secara terpisah yang masing-masing bekerja dan mengeksekusi instruksi program
Open source	Perangkat lunak yang kode sumber atau kode dasarnya dapat digunakan dan dimodifikasi oleh pengguna
Overhead	Biaya yang dikeluarkan, ongkos eksploitasi, pengeluaran tambahan
Overlays	Proses penyatuan data dari lapisan layer yang berbeda
Page table	Struktur data yang digunakan oleh MMU untuk menerjemahkan alamat memori virtual ke alamat memori fisik.
Place by calls	
Process Control Block (PCB)	Berbagai informasi lain yang diperlukan sistem operasi untuk mengontrol dan berkoordinasi dengan berbagai proses yang aktif
Programmer	Orang yang mengimplementasikan hasil rancangan aplikasi atau orang yang membuat aplikasi.
Pseudocode	Deskripsi dari algoritma pemrograman yang dituliskan secara sederhana dibandingkan dengan sintaksis bahasa pemrograman.

Random Access Memory (RAM)	Jenis penyimpanan komputer yang isinya dapat diakses dalam waktu tetap, tidak memperdulikan letak datanya dalam memori.
Read Only Memory (ROM)	Perangkat keras yang dipakai sebagai media penyimpanan data pada komputer yang sifatnya permanen
Request Edge	Set garisr berarah dari Rj ke Pi yang menunjukkan bahwa sumber daya Rj telah dialokasikan untuk memproses Pi, dan bahwa Pi saat ini memegang sumber daya Rj
Resource Allocator	Penyerahan sumber daya yang tersedia untuk berbagai penggunaan
Resource Manager	Jendela alat untuk mengimpor, membuat, mengelola, dan menggunakan resource di aplikasi
Scanner	Alat elektronik yang berfungsi untuk menggandakan berkas
Secondary Storage	Media penyimpanan data secara permanen yang di simpan untuk melayani pemrosesan data yang di lakukan oleh CPU
Semiconductor Memory	Memori computer yang terbuat dari bahan semikonduktor perangkat penyimpanan data elektronik ini biasanya diimplementasikan ke sebuah semikonduktor berbasis sirkuit terpadu (IC)
Siklus Burst	Eksekusi proses yang terdiri dari siklus CPU burst dan I/O burst. Biasanya dimulai dengan CPU burst dan kemudian diikuti oleh I/O burst, CPU burst lainnya, I/O burst lainnya dan seterusnya. Siklus ini akan terus berlanjut sampai proses dihentikan.
Singlecore	Prosesor yg hanya memiliki 1 buah inti.
Sistem *monolitic*	Struktur sistem operasi sederhana yang dilengkapi dengan operasi “dual” pelayanan {sistem call} yang diberikan oleh sistem operasi
Software	Perangkat lunak
Source Code	Sekumpulan instruksi komputer yang biasanya berbentuk teks yang berfungsi memberi perintah kerja komputer atau suatu perangkat untuk menjalankan fungsi tertentu
Store Area Network (SANs)	Jaringan berkecepatan sangat tinggi yng khusus, terdiri dari server dan penyimpanan (storage)
Streaming	Prosedur pengiriman suatu konten apakah dalam bentuk suara atau video terkompres via internet yang lalu diputar secara live tanpa pengguna perlu mengunduhnya lebih dulu
Swapping	Suatu metode pengalihan proses yang bersifat sementara dari memori utama ke suatu tempat penyimpanan sementara (disk) dan dipanggil kembali ke memori jika akan melakukan ekseskusi
System's Access Control	Pemberi ijin terhadap sebuah ojek tertentu secara spesifik

User interface	Tampilan visual sebuah produk yang menjembatani sistem dengan pengguna (user)

INDEKS

BIODATA PENULIS

Meyti Eka Apriyani, ST., MT lulus S1 Teknik Elektro Telekomunikasi Institute Teknlogi Telkom Bandung tahun 2009 lulus S2 di Program Magister Teknik Elektro Telekomunikasi pada tahun 2011.. Aktif menulis pada artikel dan berbagai jurnal ilmiah diantaranya Jurnal Informatika Polinema, Journal of Informatics, Information System, Software Engineering and Applications (INISTA) dan menjadi narasumber dalam beberapa seminar nasional..

Elok Nur Hamdana, ST., MT lulus S1 Teknik Elektro Universitas Brawijaya tahun 2009 lulus S2 di Program Magister Teknik Elektro Universitas Brawijaya pada tahun 2012. Saat ini dosen tetap Politeknik Negeri Malang pada program studi DIII Manajemen Informatika Jurusan Teknologi Informasi.

Rinanza Zulmy Alhamri, S.Kom., M.Kom. lahir di Tulungagung pada tanggal 10 April 1990 sebagai anak terakhir dari tiga bersaudara. Memperoleh gelar sarjana pada jurusan Informatika Universitas Sebelas Maret di tahun 2012 dan memperoleh gelar magister pada program studi Teknik Informatika Institut Teknologi Sepuluh Nopember di tahun 2016. Saat ini bekerja sebagai dosen tetap di Program Studi DIII Manajemen Informatika PSDKU Politeknik Negeri Malang di Kota Kediri.